12-2018

Learning Models for Discrete Optimization

Hesam Shams
University of Tennessee, hshams@vols.utk.edu

To the Graduate Council:

I am submitting herewith a dissertation written by Hesam Shams entitled "Learning Models for Discrete Optimization." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Industrial Engineering.

Oleg Shylo, Major Professor

We have read this dissertation and recommend its acceptance:

Anahita Khojandi, Michael Langston, James Ostrowski

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Learning Models for Discrete Optimization

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Hesam Shams

December 2018

Acknowledgments

I would like to express my special appreciation to my advisor Dr. Oleg Shylo who has been a great support for me. I want to thank him for his excellent cooperation and for all of the opportunities I was given to conduct my research and further my dissertation.

I would like to thank my committee members, Dr. Michael Langston, Dr. James Ostrowski, and Dr. Anahita Khojandi, for all of their guidance through this process. I want to thank them for their valuable discussion, ideas, and comments.

A special thanks to my family. Words cannot express how grateful I am to my parents for all the sacrifices they have made. I would also like to thank my beautiful and smart wife. Thank her for supporting me for everything, and especially thank her for encouraging me throughout this experience.

Abstract

We consider a class of optimization approaches that incorporate machine learning models into the algorithm structure. Our focus is on the algorithms that can learn the patterns in the search space in order to boost computational performance. The idea is to design optimization techniques that allow for computationally efficient tuning a priori. The final objective of this work is to provide efficient solvers that can be tuned for optimal performance in serial and parallel environments.

This dissertation provides a novel machine learning model based on logistic regression and describes an implementation for scheduling problems. We incorporate the proposed learning model into a well-known optimization algorithm, tabu search, and demonstrate the potential of the underlying ideas. The dissertation also establishes a new framework for comparing optimization algorithms. This framework provides a comparison of algorithms that is statistically meaningful and intuitive. Using this framework, we demonstrate that the inclusion of the logistic regression model into the tabu search method provides significant boost of its performance. Finally, we study the parallel implementation of the algorithm and evaluate the algorithm performance when more connections between threads exist.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The main objective of this dissertation is to establish optimization procedures that can learn distinctive features of an optimization problem, and can use the obtained information to improve their performance. In other words, the proposed framework is based on an integration of an optimization algorithm and a machine learning component. The field of machine learning routinely uses optimization theory and algorithms. Alternatively, the goal here is to utilize machine learning techniques to design optimization algorithms for discrete optimization.

1.1 Background and Overview

Similar ideas have recently attracted a lot of attention due to the potential computational benefits that can be obtained by embedding learning into the optimization procedures. Most machine learning applications in optimization can be divided into three different categories: models of optimal decision rules for exact solvers (e.g., branching decisions and node selection), automatic algorithm selection and parameter tuning, and optimization methods that construct machine learning models during the search process.

1.1.1 Learning models for exact solvers

A large class of exact solvers for mixed integer programming (MIP) is based on the idea of branch and bound. Two design objectives are important for these techniques: One is to avoid generation of large tree expansions without improvements to an incumbent solution, and the other is to increase the chance of proving optimality. The first can be achieved by branching variable selection, and the latter is possible by node selection on a solution tree. In the MIP literature, many studies propose approaches for node selection in branching schemes. In addition to many heuristics for branching and node selection, the recent developments in machine learning motivate researchers to explore the application of machine learning methods to guide the branching process.

The idea of using machine learning methods for branch and bound procedures was investigated by Nannicini et al. in [76]. A support vector machine (SVM) classifier was used to guess whether the probing algorithm should be applied on a given node of the branch-and-bound tree. The guessing algorithm was trained using the supervised learning methodology. The computational results showed an improvement in performance over the default branch-and-bound solvers.

In another study, Alvarez et al., [5] and its earlier version [4], have used supervised learning techniques to design a branching strategy. The authors proposed an approach in which features are extracted to characterize the branching candidates on a particular node. The training phase collected strong branching decisions, and a regression was trained to predict strong branching scores. The predictions of the model supported branching decisions in the final algorithm. The authors compared the derived branching method with other branching strategies on a set of benchmarks, and the experiments showed that the regression model had positive impact on branching.

He et al. [53] have designed a method which is able to generate efficient node searching order for the branch-and-bound procedure. The approach identifies the next node to explore during the search, with an aim to find a high-quality solution as quick, as possible. The optimality was not guaranteed because the method prunes sub-trees, which may contain the optimal solution. They trained their model by using MINLP solver software [15] on instances

from four diverse libraries and compared the approach at the test phase with the results in MINLP [15] and Gurobi [52]. The boost to solver's performance was significant according to the results.

Khalil et al. [59] have proposed a framework to support intelligent decision making during branch and bound. They trained a model to predict the rank of the strong branching scores, using an adaptive and computationally inexpensive procedure. The proposed method consists of three phases. First it derives features and node labels for the training set. Then a supervised learning is applied on the ranking function, and the final branching model is used instead of the strong branching. The approach is using a binary labeling to determine if the rank of a variable belongs to the set of high-ranked variables. The results showed that the proposed framework outperforms the default strategy of CPLEX [20]. Khalil et al. [58] also proposed a scheme to decide when to run a heuristic that tries to find a feasible solution during the branch and bound process. A logistic regression is used to learn from different instances. The features and node labels in the training phase are similar to [59]. The experiments showed strong improvement in performance, resulting from embedding the learning component into the branch and bound procedure.

If the approach includes off-line training, the training phase can be expensive computationally. Alternatively, Khalil et al. [59] proposed an approach that can be applied to instances on-the-fly. Unfortunately, the approach does not work as well as the off-line training methods. A framework was used to predict strong branching scores by Marcos Alvarez et al. [72]. This approach has no preliminary training phase and the training data is generated during the branch and bound process in contrast with the off-line learning discussed by Alvarez et al. [5] If the approximation for the strong branching scores correlated with the true scores, the algorithm starts using the approximations. The approach was not able to achieve significant improvement over the off-line framework. Khalil [57] has applied the reinforcement learning for branching in an on-line manner. In particular, the multi-arm bandit model was studied as a potential approach to tune the strategies for branching and variable selection.

The Dantzig-Wolfe decomposition method is an efficient method for solving MIP problems. When one or several decomposition structures are possible, some of them may

be more efficient for the computations. A supervised learning approach was developed by Kruber et al. [63] to choose the best possible reformulation for decomposition. The experiments on structured instances showed a meaningful improvement of the performance. In a similar study, Basso et al. [8] have used an unsupervised learning model to identify strong decomposition candidates. The experiments revealed decent accuracy of these predictions.

In [41], Fischetti and Monaci claimed that erratic behavior in the search tree is an inherent property, and instead of avoiding this behavior, they proposed a bet and run algorithm to exploit it. The algorithm evaluates several transposed versions of a model to determine which version is most likely to be solved quickly. The experimental results were compared to CPLEX's [20] default settings and showed that the proposed algorithm was faster for medium and hard instances.

A comprehensive overview of machine learning techniques in MIP solvers has been done by Lodi and Zarpellon [67]. The survey presents the problems of branching variable selection and node selection, existing early methods and their limitations, and recent approaches in which modern machine learning techniques are applied to guide decision-making in branch-and-bound procedures. Furthermore, Dilkina et al. [32] and Louveaux [69] made comments on the survey and provided some future directions for intelligent branch-and-bound in MIP.

1.1.2 Automatic algorithms configuration

When dealing with large scale problems, approximation algorithms can provide efficient solutions to discrete optimization problems in a reasonable time. Although such methods do not guarantee optimality, they can provide near-optimal solutions with a reasonable computational effort. The performance of the approximation algorithms depends on several key factors. Many such techniques have internal parameters, which require extensive tuning. The algorithm selection and parameter tuning is often done manually, using computationally expensive experimentation. Furthermore, the optimal parameters on one problem class are not necessarily optimal for another class of problem instances. Recently, researchers have employed machine learning techniques to automatically tune the parameters or to select a satisfactory algorithm for a specific problem instance.

In [55], Hutter et al. have established a framework for parameter configuration of optimization algorithms. The automatic tuning was defined as an optimization problem in the configuration space, and they considered iterated local search as the search strategy. The proposed procedure evaluated the training and testing performance using the adaptive learning strategy. They tested the approach on different benchmarks and utilized the method for tuning CPLEX solver [20]. The framework can be applied to a wide range of parameterized algorithms, including heuristics and meta-heuristics. The experiments show that the automatic parameter tuner is efficient and practical, showing successful tuning outcomes for a variety of optimization algorithms.

The design of an optimization algorithm is considered as a learning problem by Andrychowicz et al. [6], where the algorithm learns from the structure of optimization problems. Through learning, the algorithm automatically tunes its parameters to optimize its performance on a particular class of optimization problems. They trained the model to optimize a training process of the neural networks. The results show that the tuned neural optimizers outperformed optimization methods used in deep learning. A similar technique, which is called hyper configurable reactive search was proposed by Anstegui et al. [7]. The authors applied a linear regression to update parameters of a meta-heuristic. The weights of the regression were trained upfront by a configurator inspired by genetic algorithms. They tested the framework on the maximum satisfiability instances.

Hyper-heuristics include a set of approaches which aim to design optimization search automatically, and it can be described as "a search method or learning mechanism for selecting or generating heuristics to solve computational search problems" [14]. The selected heuristics are applied on the current solution, and the framework seeks to improve or generate heuristics and test the new heuristics iteratively. A critical discussion in the literature of hyper-heuristics has been presented by Burke et al. [14] to give a general review and summary on related topics.

In the algorithm selection problem, the goal is to find a solver for each instance problem with the best performance from a given set of solvers. Different approaches are studied in the literature, and machine learning techniques showed promising results. Typically, a machine learning method is applied to predict the performance of all candidate algorithms

on a specific problem, and an algorithm which is predicted to perform best is selected. The framework depends on the quality of features in differentiating instances. An overview for the algorithm selection strategy is presented by Kotthoff [62]. The author compared three different approaches to algorithm selection. The approaches were divided into regression, classification, and ranking algorithms, and the results showed that choosing among these approaches is crucial for successful applications. It is concluded that poor regression and classification models may lead to a performance that is worse than the performance can be achieved by a single good solver.

Di Liberto et al. [29] have proposed a scheme for algorithm selection which is called Dynamic Approach for Switching Heuristics (DASH). The approach applies some machine learning techniques for variable branching heuristics. First, a feature space is identified to capture aspects of the sub-problems. A portfolio including traditional branching heuristics is implemented, and all benchmark instances are divided into training and testing sets. The learning procedure starts with a clustering approach, and the problems with similar features are identified. The idea is to use the same solver on similar problems. According to the framework, at each node of the branch and bound tree, the features of the sub-problem and its nearest cluster are identified, and the procedure assigns the best branching heuristic for the selected cluster. The authors compared the framework with other static and random heuristic approaches, and the algorithm showed better performance than its counterparts.

Algorithm selection approaches use an off-line learning model to guide algorithm selection for a given problem instance from a portfolio of options. The model is trained on a variety of problem instances. A general overview of the algorithm selection with a focus on constraint satisfaction problems is studied by Bischl et al. [11]. They also have provided a benchmark library for algorithm selection which contains several algorithm selection scenarios from six different areas.

1.1.3 Learning models for search algorithms

An incorporation of learning in the optimization algorithm was pioneered by Glover [43] and Glover and Greenberg [46], where a general scheme is proposed, in which each branch is rated based on a weighted sum to choose the branch with the highest rating. The weights

can be calculated off-line by a learning procedure. Tabu search algorithm is proposed by Glover [44] in which the approach uses classification at each node to guide the search.

Daum et al. [24] have embedded a learning algorithm into a greedy search process. The proposed algorithm, search and learn (SEARN), transforms complex problems into simple classification problems, and any binary classifier can be used. The assumptions are any solution for a given problem can be decomposed into a number of independent components, and a learning technique can be applied to each component individually. However, the approach has one major drawback, it may generate infeasible solutions, even when feasible solutions are available.

A method for using machine learning inside a heuristic was proposed by Dai et al. [22], where a partial solution to a problem is updated based on a deep learning model. The authors proposed a greedy process, in which a solution is constructed incrementally. The idea here is to strengthen the machine learning branching by deep learning techniques. Extensive experiments showed that this approach is promising for learning greedy heuristics for discrete optimization problems such as vertex cover, maximum cut, and the traveling salesman problem. In a similar study, Hottung et al. [54] have applied deep neural networks for a heuristic tree search to decide on branching and pruning of tree. Their approach is called deep learning assisted heuristic tree search, and it uses a set of problems to train the heuristic to provide branching decisions for the mixed integer programming. The proposed heuristic was tested on container pre-marshalling problems, and the results showed better performance than other existing methods in the literature.

1.2 Contributions of the Dissertation

We investigate the use of machine learning (ML) techniques inside an optimization procedure. The aim of embedding ML is to design an approach which learns from the performance patterns of an optimization algorithm. The goal is to develop an efficient optimization method which is able to provide good quality solutions to the problems.

This dissertation explores the applications of the machine learning models to the design of algorithms for discrete optimization problems. The logistic regression learning model is used

to construct a guided tabu algorithm (GTA). The algorithm is tested on benchmark instances of the job shop scheduling problem. In addition, we propose an integrated framework for experimental analysis of algorithms. Finally, we outline a new methodology for modeling optimization algorithms based on the semi-Markov processes and demonstrate its potential for computationally efficient tuning.

1.3 Organization of the Dissertation

The remainder of this dissertation is organized as follows. In Chapter 2, we describe a general statistical learning model for binary optimization and establish an efficient implementation based on logistic regression. We also describe the computational performance of the model on the classic job shop scheduling problem. In Chapter 3, we design an efficient optimization algorithm that is based on the learning model from Chapter 2. In Chapter 4, we establish a framework for comparing algorithm performance, and provide a computational study of the algorithm from Chapter 3. We compare this algorithm to a similar approach that does not include a learning component, in order to quantify the value of the learning model proposed in Chapter 2. In Chapter 5, we propose Markov-based models of algorithm performance that can predict behavior in a parallel setting. We also propose a parallel implementation of the algorithm and the the experiments to compare parallel versus serial implementation of the algorithm. In Chapter 6, we discuss, the job shop scheduling problems, the applications that were used to test the proposed ideas. Finally, in Chapter 7, we discuss the results and our conclusions.

Chapter 2

Learning Models

2.1 Introduction

There is a class of algorithmic techniques in operations research and computer science that are employed to find the best option from a finite set of possibilities. Such problems are commonly referred to as combinatorial optimization problems. Many algorithms in this class operate in the following manner. The algorithm starts with some initial solution and proceeds to move from one solution to another until the best solution is found.

For example, the primal simplex algorithm for linear programming problems [23] starts by finding an initial feasible solution and iteratively transitions from one feasible solution to another using the intermediate solutions for guidance. In the case of linear programming, the algorithm continuously improves the objective, and if the improvement is not possible, the last visited solution is guaranteed to be optimal.

However, when dealing with non-linear problems, such a convenient stopping rule is not available. Many of the methods allow transitions to non-improving solutions, such as the simulated annealing method [1]. In the simulated annealing method, at every iteration, the algorithm considers a finite set of options for the future transition, and it may choose any of the non-improving options with a positive probability. In the process, similar iterative algorithms evaluate a large number of solutions, but ignore most of the information they present. Clearly, such information may provide a significant boost to performance if used efficiently, which provides the motivation for the ideas discussed in this chapter.

The main idea is to analyze the search trajectory for consistent patterns that can guide the search process. In this section we describe a statistical prediction model for binary optimization problems and explore its ability to learn the properties of the search space.

2.2 Statistical Model

A binary optimization problem can be formulated as

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{s.t. } x \in S \subset \{0,1\}^n \end{aligned} \tag{2.1}$$

Since each component of a feasible vector x is either 0 or 1, the index set $\overline{1,n} \equiv \{1,\ldots,n\}$ can be split into two classes based on an optimal solution x^* to the problem (2.1). To the first class we assign all indices for which the component value is equal to one, while the remaining indices are assigned to the second class. Formally, these classes are denoted $\mathcal{C}^1 = \{j|x_j^* = 1\}$ and $\mathcal{C}^0 = \{j|x_j^* = 0\}$, $j \in \overline{1,n}$. Clearly, there are as many such partitions as there are optimal solutions.

Here we will focus on the iterative methods, which move from one solution to another using some internal logic. As one solves the problem (2.1), some subset of feasible solutions gets discovered. Let $I_j(t)$ denote a vector of discovered information about the variable x_j that is formed by the j components of the set of feasible solutions, $x^1,\ldots,x^m \in S$, representing solutions visited by an algorithm $\mathcal{A}$ up to time t, and their corresponding objectives, $f(x_1),\ldots,f(x_m)$:

$$I_j(t) = [(x_j^1, f(x^1)), (x_j^2, f(x^2)), \ldots, (x_j^m, f(x^m))] \tag{2.2}$$

Assuming that every run of the algorithm produces a different information vector $I_j(t)$, we would like to classify $I_j(t)$ either as belonging to $\mathcal{C}_1$ or $\mathcal{C}_0$. In other words, we would like to build a prediction model M that would map vectors $I_j(t)$, $j \in \overline{1,n}$, into the interval [0,1], returning a conditional probability $Pr(j \in \mathcal{C}^1|I_j(t))$. If there are no consistent patterns in $I_j(t)$, the model should return the probabilities close to 0.5. Otherwise the probabilities may get closer to the ends of the interval [0,1], yielding predictions that can guide the search.

Clearly, the model will change with respect to the threshold time t, the time (number of iterations) that was spent to collect the information in $I_j(t)$.

2.2.1 Logistic Regression Model

The logistic regression model is a special case of a generalized linear model [42]. The logistic regression model is a predictive tool that can be used to describe the relationship between one dependent binary variable and one or more nominal, ordinal, or interval independent variables. In other words, when we have a binary output variable Y, the model predicts the conditional probability $p(x) \equiv P(Y = 1|x)$. A general formulation of logistic regression is provided by

$$\log \frac{p(x)}{1 - p(x)} = \theta_0 + \theta x \tag{2.3}$$

where, by solving the equation for p, we have the following formulation.

$$p(x; \theta_0, \theta) = \frac{e^{\theta_0 + \theta x}}{1 + e^{\theta_0 + \theta x}} = \frac{1}{1 + e^{-(\theta_0 + \theta x)}} \tag{2.4}$$

In the logistic regression model, the parameters θ_0 and θ are found by using the maximum likelihood method. The model gives us a classifier, which predicts $Y = 1$ if $p(x)$ is greater than 0.5 and predicts $Y = 0$ if the probability is less than 0.5. The decision boundary dividing the two predicted classes is the solution of $\theta_0 + \theta x = 0$. Logistic regression is one of the most widely used statistical techniques for discrete data analysis.

The hypothesis function h_θ in the logistic regression is given by the sigmoid function:

$$h_\theta(x) = \frac{1}{1 + e^{-(\theta_0 + \theta x)}} \tag{2.5}$$

When fitting (2.5) to a particular dataset (X, Y), the parameter θ can be obtained via maximum likelihood method which is proposed in [61]:

$$\theta^* = \arg\max \prod_i h_\theta(X_i)^{Y_i}(1 - h_\theta(X_i))^{1-Y_i}$$

2.2.2 Logistic Regression Model for Binary Optimization

In the optimization domain, the logistic model can be trained using the information vector $I(t)$ (2.2). Each component of $I(t)$ contains a sequence of objectives for a certain solution component. If the algorithm visited m solutions, then the total number of elements in $I(t)$ would be equal to $n \cdot m$, where n is the size of the binary solution vector. In the proposed model, all components of $I(t)$ are treated similarly, so the index of the corresponding variable does not play any role. Furthermore, the results of different runs can be merged by simply concatenating the obtained information vectors into a single information vector $I(t)$. Each component of $I(t)$ has a true label that determines whether the corresponding optimal solution value is zero or one. We will denote the vector of such ground truth labels as $L(t)$.

Consider a decomposition of $I_j(t)$ into two sequences, $I_j^1(t)$ and $I_j^0(t)$:

$$\begin{aligned} I_j^1(t) &= \{(x_j^k, f(x^k)) : (x_j^k, f(x^k)) \in I_j(t), x_j^k = 1\} \\ I_j^0(t) &= \{(x_j^k, f(x^k)) : (x_j^k, f(x^k)) \in I_j(t), x_j^k = 0\} \end{aligned} \tag{2.6}$$

Here, $I_j^1(t)$ describes all the visited objectives, where the j component was equal to one, and $I_j^0(t)$ corresponds to the objectives of solutions, where the j component was equal to zero. One of the sequences may be larger than the other, which makes it difficult to parametrize the model. To avoid this, we downsample (remove) entries in the larger sequence to match the size of the smaller sequence.

Now, the hypothesis model $h_\theta(I_j(t))$ can be defined using the sigmoid function:

$$h_\theta((I_j(t)) = \frac{1}{1 + e^G} \tag{2.7}$$

$$G = \sum_{(1, f(x_k)) \in I_j^1(t)} \theta f(x_k) - \sum_{(0, f(x_k)) \in I_j^0(t)} \theta f(x_k) \tag{2.8}$$

Notice, that there is a single parameter in this model, $\theta(t) \in \mathbb{R}^1$. There are multiple reasons for this restriction. First, assume that each variable x_j in (2.1) is substituted with

$x_j^{new} = \mathbf{1} - x_j$. Clearly, this substitution would not change the optimal objective. However, the terms in (2.8) will switch from one sum to other. Hence, by using the same parameter $\theta(t)$ for the left summation and for the right summation, we guarantee that the equivalent encodings would produce the same model. Similarly, if there was a different parameter $\theta(t)$ for each of the terms in summations, the order in which the solutions were visited would matter. However, we would like to avoid this temporal dependence by using a single parameter $\theta(t)$. Lastly, all vectors $I_j(t)$ are treated similarly to avoid reliance on the naming conventions, so the index j in $I_j(t)$ does not play any role in predictions. As a result, the data for different solution components can be merged by simply concatenating the information vectors $I_j(t)$, $j \in \overline{1,n}$, and their corresponding labels ($\mathcal{C}_1$ or $\mathcal{C}_0$) into a single data set.

Notice that the number of parameters in the model (2.7)-(2.8), depends on the number of visited solutions. The optimal value of $\theta(t)$ would change for different sizes of $I_j(t)$. To avoid this, we can project $I_j(t)$ to a smaller dimension. Next, we present one of such possible reductions using the minimum and the average function.

2.2.3 Reduced Linear Regression Model

Given $I_j^1(t)$ and $I_j^0(t)$ defined in (2.6), we can reduce them using some mapping M to $\mathbb{R}^1$. For example, such a reduction can be achieved using the minimum of average functions. Denote the resulting values by $D_j^1(t)$ and $D_j^0(t)$:

$$D_j^1(t) = M(\{f(x_j^k) | (x_j^k, f(x^k)) \in I_j(t), x_j^k = 1\})$$
$$D_j^0(t) = M(\{f(x_j^k) | (x_j^k, f(x^k)) \in I_j(t), x_j^k = 0\})$$

This provides us with the reduced information vector $I_j^R(t) = [D_j^1(t), D_j^0(t)] \in \mathbb{R}^2$, $I_j(t)$ is mapped to a vector in $\mathbb{R}^2$. Clearly, instead of storing $I_j(t)$ in the memory for these calculations, $I_j^R(t)$ should be updated directly every time the algorithm finds a new feasible solution.

The corresponding logistic regression model can be described using the following hypothesis function h_θ.

$$h_\theta(D_j^1(t), D_j^0(t)) = \frac{1}{1+e^G} \tag{2.9}$$

$$G = \theta D_j^1(t) - \theta D_j^0(t) \tag{2.10}$$

To clarify the problem, consider the snapshot of the training data in Table 2.1. Two predictors, $D^1(t)$ and $D^0(t)$ are used to predict whether the corresponding component is equal to zero or one, with the ground truth values in column opt. To collect the training data for columns $D^1(t)$ and $D^0(t)$, the algorithm runs repeatedly for time t and reports the best objective for each solution component. To collect the training data for columns $D^1(t)$ and $D^0(t)$, the algorithm runs repeatedly for time t and reports the best objective for each solution component. Note that an algorithm may never encounter certain solution components with a value of zero (or one). In this case, we omit the corresponding rows from the data set, and the model prediction is fixed to 0.5 for those components.

Table 2.1: An example of the training data for the logistic regression model.

$D^1(t)$	$D^0(t)$	opt
1395	1366	0
1368	1400	1
1366	1438	1
1373	1366	0
1379	1365	0
1365	1389	1

In practice, the model (2.9) should be tuned on a representative set of optimization problems, for which optimal or high-quality reference solutions are already established. Then the tuned model can be applied to new problem instances without known solutions, assuming that the new instances are "similar" to the problem set used to parametrize the model.

In the following, we explore the empirical distribution of the optimal regression parameter $\theta(t)$ for different values of the time threshold t for a set of job shop scheduling instances [56], and apply an iterative algorithm based on the tabu search [79] to collect the training data. The choice of the tabu search is motivated by the wide domain of successful applications in the literature. While the job shop scheduling problem captures many complexities common to binary optimization domains.

2.3 Computational Results

Note that the optimal regression parameter $\theta(t)$ in (2.9) depends on the specific training dataset, which in our case is generated by an optimization algorithm solving a problem instance for a fixed number of iterations. To collect the data we need to chose an optimization algorithm, specific problem instances to solve, and a time threshold for data collection. As a proof of concept, the proposed framework was tested using the tabu search algorithm [48] applied to the job shop scheduling problem. As a general purpose technique, the tabu method does not use any problem specific features of the job shop scheduling formulation and mainly relies on exploring the search space through a sequence of local moves. Hence, the following discussion should be viewed as a proof of concept, and is neither limited to the job shop scheduling problem, nor to the tabu search. The same methodology can be applied to any iterative binary optimization algorithm.

The choice of an application and solver is arbitrary, but it is motivated by the wide domain of problems reported in the literature that employed the tabu search method. In many applications the tabu method is widely accepted as one of the best performing solvers [21, 48]. As a general purpose technique, the tabu method does not use any problem specific features of the job shop scheduling formulation and mainly relies on exploring the search space through a sequence of local moves based on a 1-flip operator (switching a single solution component x_i to $1 - x_i$). While the computational data does not transfer to other combinatorial problems, the methodology provides a general purpose framework that can be applied for a large domain of applications. This experimental design does not limit the scope of potential applications neither to scheduling nor to the tabu search method. In particular, we use the tabu algorithm proposed in [78], which is widely accepted as one of the best computational approaches for the job shop scheduling problem.

2.3.1 Experimental Setup

We consider four classes of the job shop scheduling instances which are grouped by the problem size: ta11-20, ta21-30, ta31-ta40 and ta41-ta50. The detailed description of these problem instances is provided in Chapter 6. In the optimization field, an ideal algorithm

has to satisfy three requirements according to [92]: (1) it should be able to find an optimal solution (2) in polynomial time (3) for any problem instance. However, there is a large class of problems for which such ideal solvers are not available, *i.e.*, a class of NP-hard problems. For many of these problems, often inspired by practical applications, it is reasonable to relax the first requirement. An algorithm that satisfies the remaining conditions is commonly referred to as an approximate algorithm. A good approximation algorithm is able to provide high quality solutions (not necessarily optimal) for a wide variety of problem instances in a reasonable amount of time. Most practical problems in discrete optimization are NP-hard which stimulated the develop of approximate techniques.

When considering successful applications, the tabu search method [44] is arguably one of the best standalone optimization approaches among those based on local search. Local search provides a framework, which transforms one solution into another through modification of their constituent attributes. Tabu search employs a short-term memory prohibition mechanism, a rule that prevents revisiting of solution attributes recently removed from the current solution. Less commonly, tabu restrictions inhibit removal of attributes that were recently introduced into the current solution. In general, these two types of restrictions lead to different search trajectories and might be employed in parallel, however in the case of 0-1 optimization problems, they are equivalent [44]. Through inhibition mechanisms and by enabling non-improving solution attributes, the tabu search method provides an almost effortless escape from local minima together with efficient exploration of alternative solutions.

Typically, when a certain attribute enters a list of prohibited attributes, the tabu list, it will remain there for a fixed number of iterations determined by a *tabu tenure* parameter. Most tabu search implementations adopt a single tabu tenure parameter for each of the solution attributes, which is often defined as a function of problem size and might be dynamically adjusted to avoid cycling effects [9]. The attribute-dependent tenures, where each solution attribute is assigned a separate tabu tenure value, has been also identified in earlier publications [47, 45]. However, the attribute-dependent tenures mainly focused on the variability with respect to restrictive powers of different move attributes [47], with an emphasis being placed on an idea that when using the same tabu tenure for all solution

components, prohibition of certain solution attributes might have a stronger impact on search process than prohibition of the others.

Many optimization approaches rely on the tabu method, but often utilize additional mechanisms for diversification and intensification of the search. For example, multi-start tabu strategies repeatedly launch the tabu search procedure using different initial solutions. In the path-relinking framework, one collects a set of diverse high-quality solutions, the elite set, constructs paths between them, and explores the neighborhoods of the intermediate solutions using local search or tabu search procedures. However, when implementing a path-relinking algorithm, there are many questions that are not easy to answer: what is the optimal size of the elite set, how much time should be spend constructing the elite set versus exploring the paths between them.

Algorithm 2.1 describes our implementation of the tabu search method. The vector $tabuExp$ keeps track of the tabu status of solution components. For example, if $tabuExp[5] = 100$, then the fifth component of the solution vector is considered tabu as long as the iteration counter is less than 100. If there are multiple solution components involved, the tabu status is defined by the component with the largest value of $tabuExp$. The algorithm runs for a fixed number of epochs, each consisting of $niters$ iterations. At the beginning of each epoch, the algorithm reverts to the current best known solution (Algorithm 2.1, line 5). The duration of the tabu tenure is set to a random integer in the interval $[T_{min}, T_{max}]$ to prevent cycling. The algorithm scans through all the solutions in the neighborhood of the current solution x^0 (Algorithm 2.1, lines 9-13) to identify the set of prohibited (tabu) solutions. The non-tabu solution with the best objective value is chosen as the next incumbent solution, and using the random tie-breaking if necessary. If all the solutions in the neighborhood are prohibited, then the algorithm chooses the solution with the earliest expiration of its tabu status. The best found solution is returned after the fixed number of search epochs.

Note that the algorithm has only two components which are problem specific: the neighborhood $N(x)$ and the specific objective function $f(x)$. In our case, the search is based on the N4-neighborhood for the job shop scheduling problem proposed in [12], which is based on the concept of a critical path. The N4-neighborhood consists of solutions obtained by moving operations either to the beginning or to the end of their critical blocks.

```
1: procedure TabuSearch(T_min, T_max, nepochs, niters)
2:     generate a random solution x^min
3:     tabuExp(j) = 0, j = 1, ..., |x^min|;                ▷ initialize tabu status expiration
4:     for epoch = 1 to nepochs do
5:         x^0 = x^min
6:         for iter = 1 to niters do
7:             tenure ← generate a random integer from [T_min, T_max]
8:             for x in N(x^0) do
9:                 expir(x) = iter
10:                for j in {j : x_j^0 ≠ x_j} do
11:                    expir(x) = max(tabuExp(j), expir(x))
12:                if expir(x) > iter then
13:                    TabuSet = TabuSet ∪ x
14:            NonTabuSet = N(x^0) - TabuSet
15:            if NonTabuSet ≠ ∅ then
16:                x^new = arg min{f(x) : x ∈ NonTabuSet};          ▷ best non-tabu solution
17:            else
18:                x^new = arg min{expir(x) : x ∈ N(x^0)};          ▷ tabu solution with earliest
   expiration
19:            for j in {j : x_j^new ≠ x_j^0} do; ▷ only look at the components that have changed
20:                tabuExp(j) = iter + tenure
21:            x^0 = x^new
22:            if [f(x^0) < f(x^min)] then
23:                x^min = x^0
    return x^min
```

Algorithm 2.1: Pseudocode of the multi-start tabu search algorithm.

2.3.2 Computational Results

The collected data for problem ta11 consisted of 342,000 rows and 3 columns ($D_j^1(t)$, $D_j^0(t)$ and $L(t)$). This dataset was randomly split into the training, validation and testing sets, with the number of samples equal to 152000, 76000 and 114000, correspondingly. The percentage of zero labels in each of the data sets was 0.45 (validation set), 0.46 (training set) and 0.46 (testing set).

The model was trained using the data in the testing set, and the summary of the predictions for the testing set are presented in Tables 2.2 and 2.3. The model correctly predicts 93% of the labels, or in optimization terms, the model can accurately predict 93% of the optimal solution components. An example of the predicted probabilities is presented in Figure 2.1a (minimum reduction) and Figure 2.1b (average reduction).

Table 2.2: Confusion matrix for the logisitic model predictions on the testing set of data.

	pred=0	pred=1
real=0	52723	2982
real=1	4531	53764

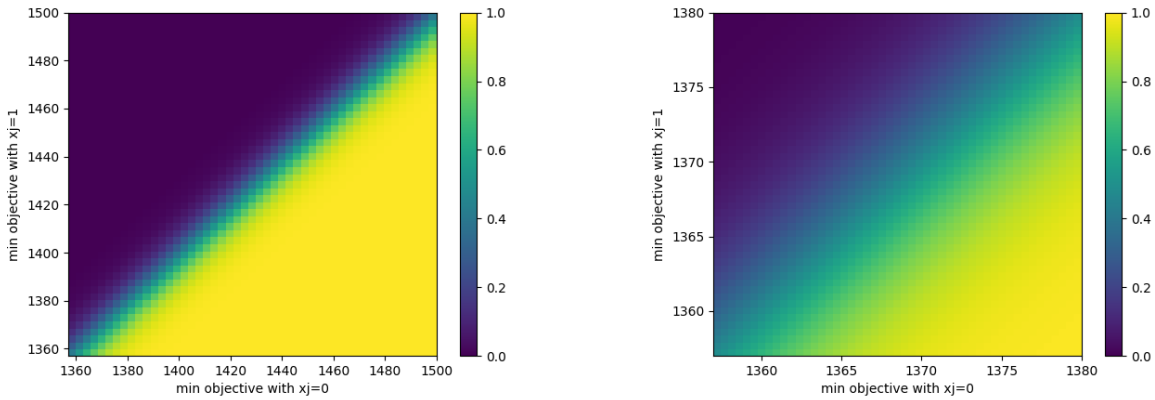
Table 2.3: Normalized confusion matrix for the logisitic model predictions on the testing set of data.

	pred=0	pred=1
real=0	0.95	0.05
real=1	0.08	0.92

In the first set of experiments we investigated the predictive potential of the logistic regression model (2.9). The tabu search was used to solve Taillard's instances ta11-ta50. In total, we performed 20 runs, each run was limited to $nepochs = 200$ and each epoch consisted of $niter = 300,000$ iterations. The lower tenure range parameter T_{min} was set to 5 and T_{max} was set to 11.

In every run we continuously updated the values D_j^1 and D_j^0 for each solution component j as described in Section 2.2.3. Remember that these values are the best objectives found when the component j is equal to one and zero respectively, so each update to the vectors D_j^1 and D_j^0 takes $O(|x|)$ steps, where $|x|$ is the size of the binary solution vector. In order to decrease

the time spent updating these structures, the updates were performed every 100 iterations or whenever the best known solution was improved. To provide the target/output values in the training data, the record solutions from http://optimizizer.com/jobshop.php were used.



(a) Based on the minimum objective for each solution component.

(b) Based on the average objective for each solution component.

Figure 2.1: Heatmap of predictions from the logistic model.

At each epoch the algorithm provided a table of training data similar in structure to the one presented in Table 2.1. Using this table, at the end of each epoch we generated the maximum likelihood estimator for the logistic regression parameter θ using the model (2.9). Figure 2.2 shows the average θ values and the corresponding accuracy of the logistic regression model over 20 runs along with the corresponding 95% confidence intervals. The accuracy plots confirm that the best objective values D^1 and D^0 generated by the tabu search are relatively good predictors of the optimal solution values, with the average accuracy between 80% and 95%. Also it is clear that the optimal regression parameter, as well as the model accuracy tends to increase over time, until it reaches a saturation point when the algorithm is not able to find any better solutions.

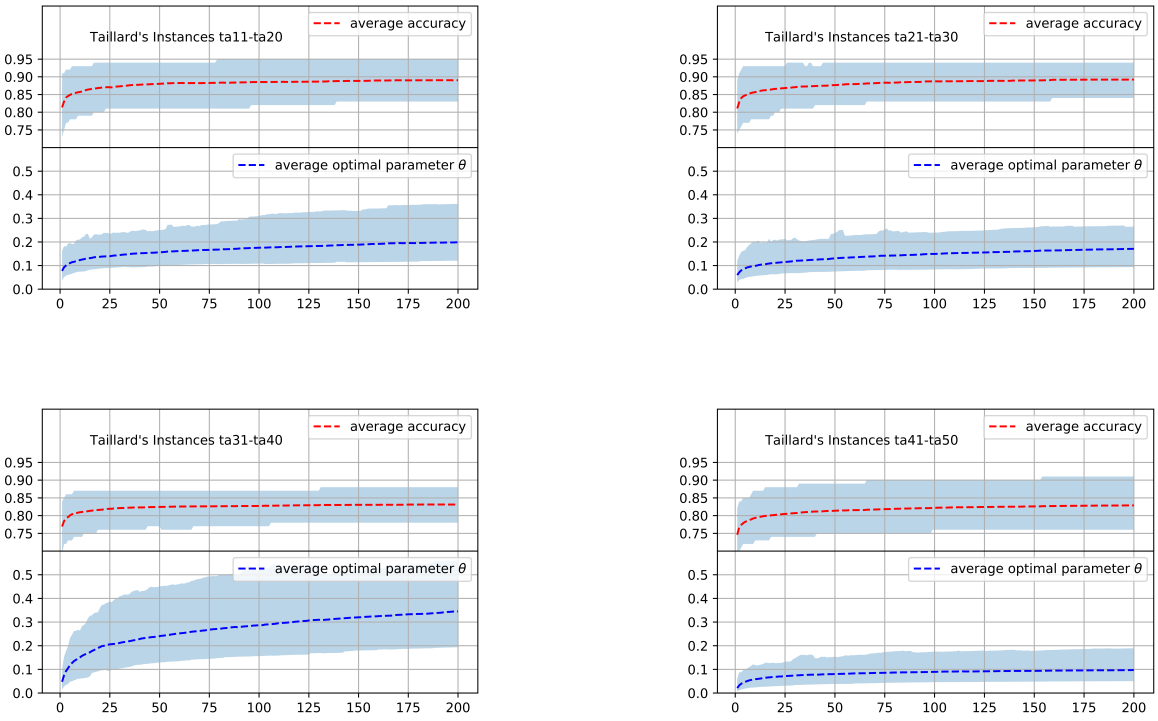


Figure 2.2: Optimal regression parameter θ and the corresponding accuracy of the logistic regression model (2.9) on the set of Taillard's benchmarks ta11-ta50.

2.3.3 Conclusions

The proposed logistic regression model achieves high accuracy for the tabu algorithm on the considered class of job shop scheduling problems. The optimal parameter θ changes with respect to the time used to collect the information, but the range of the optimal values is stable across different problem instances. Hence, it is possible to find an interval $[\theta_{min}, \theta_{max}]$ that contains all optimal values for a given set of problem instances and time threshold t. Our working hypothesis is that this interval would provide a good estimate for the location of optimal θ values for the problem instance that were not used to train the model. In some sense, the interval $[\theta_{min}, \theta_{max}]$ provides a prediction for what is optimal for a class of scheduling problems.

Importantly, in order to find an optimal value of θ for a given problem, one needs to know an optimal solution of that problem. Clearly, when considering an optimization problem, which has not been previously solved, its optimal solution is not available. However, we can use the range $[\theta_{min}, \theta_{max}]$ estimated from the testing problems as a predictor for the location of the optimal logistic parameter of the new problem instance. Given a grid of points from the interval $[\theta_{min}, \theta_{max}]$, one of the points will be close to the optimal value. This property can be used to develop an algorithm that can use the logistic model to guide the search process.

Consider a sequence of values $\theta_1 < \theta_2 < \ldots < \theta_T$ from the interval $[\theta_{min}, \theta_{max}]$. From the previous discussion we assume that the optimal value θ^{opt} is close to one of these values. The algorithm will start with a parameter θ_1 and run for a fixed amount of time before switching to θ_2. Next, the algorithm would switch to θ_3, and so on. As one of the θ values is close to θ^{opt}, we are guaranteed that the algorithm parameters would be close to the optimal settings at some point of its execution. This idea is similar to the temperature schedule of the simulated annealing method [1]. But unlike the simulated annealing procedure, the proposed scheme draws its logic from the theory of maximum likelihood estimators for the logistic regression.

Chapter 3

Guided Tabu Algorithm

3.1 Introduction

In the previous chapter, we outlined the regression model for binary optimization and provided the case study for the job shop scheduling problems and the tabu search method. Clearly, when considering a new problem, the tuned model constructed in that study has limited value, as it requires an optimal solution of the new problem to find an optimal parameter value. However, the model includes only one parameter. On the other hand, there exist significant amount of data to train the simple mode.

In order to train the proposed logistic regression models, an optimal solution to the problem is required. Experimental analysis showed that the proposed model can be quite accurate in practice.

Since the logistic regression is general in design, it is not restricted to a single technique and is easy to be added to other techniques. So, we decided to use it with tabu search. Although, tabu is a general solver, researchers applies different approaches to use the explored information. The embedding of logistic regression model is another alternative to use the information.

We propose a method to embed the learning model inside the generic tabu search proposed in Algorithm 2.1. Each component of a solution vector is assigned a separate tabu tenure value that is dynamically updated and depends on previously found solutions. To define the values for tabu tenures, we use a logistic regression model as is described in

Chapter 2. The proposed algorithm is inspired by the Global Equilibrium Search method [85] and is described in the followings.

3.2 Description of the approach

3.2.1 Main Idea

Consider a binary optimization problem as follows:

$$\begin{aligned} &\min f(x) \\ &\text{s.t. } x \in S \subset \{0,1\}^n \end{aligned} \tag{3.1}$$

In the simplest form, the tabu search algorithm iteratively moves from one solution to another using the values of the corresponding objective values for guidance. Given a current solution x, at each iteration the algorithm moves to one of the solutions in its neighborhood $N(x)$, however the tabu search method prohibits some of the solutions in $N(x)$. Suppose that $latestChange(j)$ is the latest iteration when the solution component j changed its value, then any solution in $N(x)$ that differs from x in the component j is prohibited until the iteration number $latestChange(j) + tenure$, where $tenure$ defines a length of the tabu period. There are many variations of tabu search implementations, but the idea is similar: prohibit changes in components that were recently modified. Typically, the best non-tabu solution in $N(x)$ is chosen as a next current solution, and after that the process repeats.

In the current chapter, we explore an approach that uses the tabu prohibition mechanism both for escaping from local minima, and for guiding the search to promising solution areas. Instead of a single tabu parameter, each solution component is assigned its own tabu parameter tabu_j that is dynamically updated during the search. By assigning large values to tabu_j, the algorithm attempts to preserve the current value of the x_j, while small tabu_j will indicate that the component x_j can be modified at a faster pace. For example, if we wish to guide the tabu search to a specified solution x^*, we can use a standard tabu search procedure, but whenever x_j takes the same value as x_j^*, we would set tabu_j to T^U, and set it to T^L if the new x_j is different from x_j^*, where $T^U > T^L$. If the neighborhood is

connected (any solution can be reached from any other solution), then an appropriate choice of T^L and T^U will guarantee the convergence.

3.2.2 Long-term Memory

To accumulate information about the search space, we will use a logistic regression model from Chapter 2. Given $I_j^1(t)$ and $I_j^0(t)$ (defined in (2.2), we reduce them using the minimum function to $\mathbb{R}^1$. Denote the results of such reduction by $D_j^1(t)$ and $D_j^0(t)$.

$$D_j^1(t) = \min(\{f(x_j^k) | (x_j^k, f(x^k)) \in I_j(t), x_j^k = 1\})$$

and

$$D_j^0(t) = \min(\{f(x_j^k) | (x_j^k, f(x^k)) \in I_j(t), x_j^k = 0\})$$

This provides with the reduced information vector $I_j^R(t) = [D_j^1(t), D_j^0(t)] \in \mathbb{R}^2$. Hence, we reduce $I_j(t)$ to a vector in $\mathbb{R}^2$. Clearly, instead of storing $I_j(t)$ in the memory for these calculations, $I_j^R(t)$ should be updated directly every time the algorithm finds a new feasible solution.

The corresponding logistic regression model can be described using the following hypodissertation function h_θ.

$$h_\theta(D_j^1(t), D_j^0(t)) = \frac{1}{1 + e^G} \tag{3.2}$$

$$G = \theta D_j^1(t) - \theta D_j^0(t) \tag{3.3}$$

To guarantee that the expressions in (3.2) and (3.3) are well-defined, we initialize each information vector using some large constant f_{init} as follows.

$$I_j^{init}(t) = \{(1, f_{init}), (0, f_{init})\} \tag{3.4}$$

The model will provide the predictions of optimal value for each solution component. The probability of component j in the optimal solution equals to 1 is calculated as follows:

$$\tilde{p}_j(\theta) \equiv \frac{1}{1 + e^{\theta D_j^1(t) - \theta D_j^0(t)}} \approx P(x_j^* = 1) \tag{3.5}$$

3.2.3 Dynamic tabu search tenure

In the proposed approach, the logistic regression model from Chapter 2 defines dynamic tabu tenures. Whenever x_j is modified, we compare its new value to the current best solution x_j^{best}. If the probability in (3.5) is close to 1 or 0 and the new value is the same as x_j^{best}, then we assign a large tenure value to x_j. Otherwise, we want to enforce a faster rate of change for x_j, so we assign a smaller tenure value. Next, we define a function that links probabilities to tabu tenures.

3.2.4 Quadratic Tenure Function

After every local search transition, the tenure for each component that has changed is determined by the following quadratic function.

$$\text{tabu}_j(\tilde{p}_j(\theta)) = \begin{cases} 4(T^U - T^L)\tilde{p}_j(\theta)^2 - 4(T^U - T^L)\tilde{p}_j(\theta) + T^U & x_j = x_j^{best} \\ T^L & x_j \neq x_j^{best} \end{cases} \tag{3.6}$$

where an interval $[T^L, T^U]$ defines a range of possible tenure values. The coefficients of this quadratic function are chosen to satisfy the following equalities.

$$\text{tabu}_j(\tilde{p}_j(\theta)) = T^U \text{ if } \tilde{p}_j(\theta) = 1 \tag{3.7}$$

$$\text{tabu}_j(\tilde{p}_j(\theta)) = T^U \text{ if } \tilde{p}_j(\theta) = 0 \tag{3.8}$$

$$\text{tabu}_j(\tilde{p}_j(\theta)) = T^L \text{ if } \tilde{p}_j(\theta) = 0.5 \tag{3.9}$$

If the component j is set to a different value other than the best known solution, then the variable j is assigned a low tenure value T^L. Otherwise, the tenure value is a quadratic function of the probabilities provided by the logistic regression: the closer probability $\tilde{p}_j$ is

to 1 or 0, the larger is the value of the assigned tabu tenure, with the maximum possible value of T^U.

3.2.5 Other possible choices for the tenure function

Sigmoid Tenure Function

Similarly to the quadratic function, the assigned tenures belong to the interval $[T^L, T^U]$. Whenever a solution component x_j is modified, its tenure is determined by the function as follows.

$$\text{tabu}_j(\tilde{p}_j(\theta)) = \begin{cases} \frac{2T^U + T^L \exp(\alpha\tilde{p}_j(\theta)) - T^L}{1+\exp(\alpha\tilde{p}_j(\theta))} & x_j = x_j^{best}, \tilde{p}_j(\theta) \leq 0.5 \\ \frac{2T^U + T^L \exp(\alpha(1-\tilde{p}_j(\theta))) - T^L}{1+\exp(\alpha(1-\tilde{p}_j(\theta)))} & x_j = x_j^{best}, \tilde{p}_j(\theta) > 0.5 \\ T^L & x_j \neq x_j^{best} \end{cases} \tag{3.10}$$

where parameter $\alpha > 0$ defines the steepness of the function, and it should be chosen to satisfy the condition in (3.11).

$$\text{tabu}_j(\tilde{p}_j(\theta)) \approx T^L \qquad \text{if} \quad \tilde{p}_j(\theta) = 0.5 \tag{3.11}$$

This function equals to T^U when $\tilde{p}_j(\theta)$ equals to 1 and 0 as is shown in the followings.

$$\begin{aligned} &\text{tabu}_j(\tilde{p}_j(\theta)) = T^U \text{ if } \tilde{p}_j(\theta) = 1 \\ &\text{tabu}_j(\tilde{p}_j(\theta)) = T^U \text{ if } \tilde{p}_j(\theta) = 0 \end{aligned} \tag{3.12}$$

Unbounded Tenure Function

The tenure of the modified solution component x_j is computed as follows:

$$\text{tabu}_j(\tilde{p}_j(\theta)) = \begin{cases} T^L \frac{1-\tilde{p}_j(\theta))}{\tilde{p}_j(\theta)} & x_j = x_j^{best}, \tilde{p}_j(\theta) \leq 0.5 \\ T^L \frac{\tilde{p}_j(\theta))}{1-\tilde{p}_j(\theta)} & x_j = x_j^{best}, \tilde{p}_j(\theta) > 0.5 \\ T^L & x_j \neq x_j^{best} \end{cases} \tag{3.13}$$

The plots of tenure variations by different tenure functions are shown in Figure 3.1.

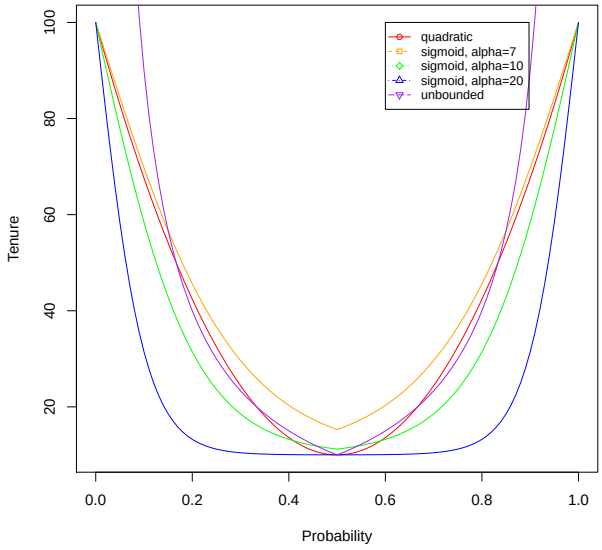


Figure 3.1: Tenure as a function of approximation probabilities, $T^U = 100$ and $T^L = 10$.

3.2.6 Optimal Value for Regression Model

To tune the regression model for a specific problem instance, one needs to know an optimal or high-quality solution to that instance. When solving a new problem, the regression parameter has to be re-calibrated. Clearly, an optimal solution of the new problem instance is not available for such re-calibration, unless we solve the problem. Is is possible to use the logistic model to boost the search process?

From Figure 2.2, notice that the distribution of average optimal θ values is rather robust: $\theta \in [0, 0.5]$ and the standard deviation less than 0.1. Hence, when solving the problems ta11-ta50, the optimal θ belongs to [0,1.0] interval with high probability. Instead of finding the optimal regression parameter value, we generate a sequence of "potential" optimal values $0 = \theta_0 \leq \theta_1 \leq \ldots \leq \theta_K = 1.0$, and scan through them one by one, generating model predictions. If the interval discretization is fine enough and a problem instance is similar to ta11-ta50, one of the θ values will be close to the optimal value and the regression model will produce optimal predictions based on the available data.

Next, we outline the Guided Tabu Algorithm (GTA), which combines the standard tabu method and the logistic regression model described in the previous section. The proposed algorithm is inspired by the Global Equilibrium Search method [85].

3.2.7 Algorithm Description

Instead of using a single tabu tenure parameter, each component of a solution vector is assigned a separate tabu expiration parameter $tabuExp(j)$, which provides the iteration at which the component's tabu status expires. These expiration times are updated whenever the component changes its value. By assigning large expiration times $tabuExp(j)$ the algorithm attempts to preserve the current value of x_j, while small expiration times $tabuExp(j)$ will induce a faster rate of change for x_j.

The attribute-dependent tenures, where each solution attribute is assigned a separate tabu tenure value, has been also identified in earlier publications [47, 45]. However, previous discussions of the attribute-dependent tenures mainly focused on the variability with respect to restrictive powers of different move attributes [47], with an emphasis being placed on an idea that when using the same tabu tenure for all solution components, prohibition of certain solution attributes might have a stronger impact on search process than prohibition of the others.

From the previous section, the reduced logistic regression model uses two values, D_j^1 and D_j^0, to predict a probability that $x_j^* = 1$ in an optimal solution x^*:

$$p_j \equiv \frac{1}{1 + e^{\theta D_j^1(t) - \theta D_j^0(t)}} \approx P(x_j^* = 1).$$

Notice that every variable with a tabu status can be in one of two states: $x_j = 1$ or $x_j = 0$. By setting distinct tenure values for each of these states, we can assure that the ratio of overall time spend in each of them is proportional to p_j and $1 - p_j$. For example, this is achieved by the following mapping:

$$T_j = \begin{cases} T^L \cdot \frac{p_j}{\max(1-p_j,\epsilon)} & \text{if } x_j^{new} = 1 \text{ and } p_j \geq 0.5, \\ T^L & \text{if } x_j^{new} = 1 \text{ and } p_j < 0.5, \\ T^L \cdot \frac{1-p_j}{\max(p_j,\epsilon)} & \text{if } x_j^{new} = 0 \text{ and } p_j \leq 0.5, \\ T^L & \text{if } x_j^{new} = 0 \text{ and } p_j > 0.5. \end{cases}$$

This mapping assures that if after the local move, the new value of the variable x_j^{new} is not consistent with the model prediction p_j, then it will be assigned a standard tenure value of T^L. Otherwise, the variable is assigned a larger tenure proportionally to p_j to promote a slower rate of change. The parameter ϵ is a small constant that prevents division by zero (e.g., we use $\epsilon = 10^{-4}$ in all the computations). Algorithm 3.1 provides a description of the **Guided Tabu Algorithm** (GTA).

procedure GUIDEDTABUSEARCH(x^{min} – current best solution, θ_{min}, θ_{max})
  $tabuExp(j) = 0$, $j = 1, \ldots, |x^0|$; ▷ initialize tabu status expiration
  $x^0 = x^{min}$
  for $epoch = 1$ to $nepochs$ **do**
    $\theta = \theta_{min} * e^{(\log(\theta_{max}) - \log(\theta_{min})) \cdot epoch/nepochs}$
    for $iter = 1$ to $niters$ **do**
      for x in $N(x^0)$ **do**
        $expir(x) = iter$
        for j in $\{j : x_j^0 \neq x_j\}$ **do**
          $expir(x) = \max(tabuExp(j), expir(x))$
        if $expir(x) > iter$ **then**
          TabuSet = TabuSet $\cup$ x
      NonTabuSet = $N(x^0)$ - TabuSet
      if NonTabuSet $\neq \emptyset$ **then**
        $x^{new} = \arg\min\{f(x) : x \in \text{NonTabuSet}\}$ ▷ best non-tabu solution
      else
        $x^{new} = \arg\min\{expir(x) : x \in N(x^0)\}$ ▷ tabu solution with earliest expiration
      if $iter \bmod d = 0$ or $f(x^0) < f(x^{min})$ **then**
        update vectors D^1, D^0
        calculate probabilities $p_j = 1/\left(1 + e^{\theta D_j^1(t) - \theta D_j^0(t)}\right)$, $j = 1, \ldots, |x^0|$;
      for j in $\{j : x_j^{new} \neq x_j^0\}$ **do** ▷ only look at the components that have changed
        if ($x_j^{new} = 1$ and $p_j \geq 0.5$) or ($x_j^{new} = 0$ and $p_j \leq 0.5$) **then**
          $tabuExp(j) = iter + tenure \cdot \frac{\max(p[j], 1-p[j])}{\max[\epsilon, \min(p_j, 1-p[j])]}$
        else
          $tabuExp(j) = iter + tenure$
      $x^0 = x^{new}$
      if $[f(x^0) < f(x^{min})]$ **then**
        $x^{min} = x^0$
  return x^{min}

Algorithm 3.1: Pseudocode of the guided tabu search algorithm.

Chapter 4

Experimental Analysis of Algorithms

4.1 Introduction

This chapter describes an integrated framework for comparing optimization algorithms based on the concept of probability dominance. This approach provides an assessment of algorithms performance which is intuitive and statistically sound. The framework includes parametric and non-parametric models which are able to provide the probability of reaching better solutions in a pair-wise comparison of algorithms with its confidence interval on a specific time. Furthermore, these probabilities can be plotted over a period of time to see the trends of efficacy. We also implemented the framework on random sets of solutions to see the coverage probability of different models. This framework has been coded in `Python` which is called `algo-plot` package and it can be widely used as a tool for algorithm design and development in the operations research.

One of the most important steps in design of algorithms is finding an approach to evaluate the performance. A number of research have been studied on analysis of algorithms to examine the efficiency which can be divided into three main categories: the worst case analysis, the average case analysis, and the experimental analysis.

The worst case analysis focuses on the rigorous theoretical guarantees of the algorithm performance (e.g., run time) in the worst possible scenario. For some problems, the worst case scenarios are quite common, but often it is not the case, which leads to overly pessimistic predictions for its performance. The average case analysis considers the expected

performance for a particular distribution of the input data. The common difficulty with this approach is to identify the average-case input that is well-justified for applications.

The experimental analysis of the algorithms is very common for the modern optimization techniques. While the worst-case or the average-case approaches often provide solid theoretical guarantees of the algorithmic performance, they cannot provide a general overview of performances for practical reasons. The worst-case guarantees are usually too pessimistic (e.g., in the worst case scenario, the simplex algorithm may visit 2^n vertices for the problem of size n), and the tractable average-case derivations may require distributional assumptions that are far from reality.

The objective of the experimental analysis is typically to highlight the strengths and weaknesses of an algorithm. Such analysis is commonly accompanied by the comparison with the previously known methods, which is sometimes called a *horse race* analysis, as it focuses on showing that the algorithm dominates the existing competition. For example, Mittelmann [73] runs commercial and open source solvers (such as CPLEX [20] or Gurobi [52]) on a set of the benchmark instances and maintained a comparison analysis over solvers.

The focus of this chapter is on the experimental analysis of the algorithms. The common approach to experimental analysis of an algorithm involves a set of a well-established benchmark problems. The algorithm is applied to these problems, and the results are reported in the form of tables. The main problem with these experimental evaluations is a lack of a common experimental design which usually renders a meaningful comparison impossible. For example, the exact algorithms would often report run times till optimal solutions are obtained. If the problems are hard enough, the algorithm might not find the optimal solution within a certain time threshold, prompting the reports of the optimality gaps or percentage of problems solved to optimality. Clearly, the choice of the time threshold would heavily impact the reported results. The algorithm that dominates the field with one hour of computing time, might perform poorly if two hours of computing time are considered. As the researchers are an interested party, they might be inclined to choose the threshold that favors their algorithm. Furthermore, the table results might not clearly indicate the superiority of one algorithm over another, as the method would look better on some problems, but not on the others.

In this chapter, we present an integrated framework for comparing algorithms that alleviates some of the drawbacks discussed above. First, we present a review of some common approaches for algorithm evaluations.

4.2 Literature Review

Since it is not possible to test the algorithm performance on every instance of NP-hard problems, the research community often identifies a comprehensive set of sample problems. These problems are often match the size of practical problems, and are commonly considered difficult for the state-of-the-art optimization methods. For example, such collections of benchmark exist for the scheduling problems (Taillard's instances) [88], traveling salesman problems (Reinelt's instances) [81], vehicle routing problems (Golden's benchmark) [49]. Benchmarking process is reviewed comprehensively by Beiranvand et al. [10] and challenges of different approaches are discussed.

After choosing the benchmark set, the algorithm is evaluated with respect to the run time and solution quality. Different authors have applied different experimental designs for such evaluations. The common approach is to record the best objective value or the best lower bound obtained within a fixed time budget for every instance in the benchmark set similar to [74, 75].

In the case of the randomized algorithms, the authors often count the number of time the algorithm finds an optimal solution (success rate) for each benchmark, for instance [96]. The performance report may include the average optimality gaps across a fixed number of runs. This approach may not be practical, as it requires a benchmark which can be solved exactly in order to calculate meaningful success rates and average optimality gaps. Another metric for evaluation of the algorithm performance is the number of function evaluations, for instances [77, 68]. The function evaluation can be fitness evaluation in evolutionary algorithms which the comparison is similar to [34, 87]. The issue of this metric is that it can only be applied to the methods based on local search based methods. The comparison between algorithms that use different neighborhood structures is not possible, and it may be difficult to translate the reported data to the run time values.

Another class of widely used approaches is based on statistical methods, such as analysis of variance (ANOVA) or hypothesis testing. For example average relative percentage deviation (RDP) has been used as the performance measure in [80, 40]. Mean-value of the objectives values and its standard deviation was proposed to measure the algorithm performance in [18]. A broad review on similar approaches can be found in [28] and a review on developments of comparison methods of evolutionary algorithms has been done by Dimopoulos and Zalzala [33]. In a comparable study, Carrano et al. [16] proposed a multiple criteria for comparing algorithms and construct an algorithm ranking table. The common feature of these approaches is that all provide numerous tables filled with different scaled numbers which do not give a general overview to the algorithm performance and can be quite confusing. In order to resolve this confusion, visual comparison methods can provide a useful tool for a clear presentation of the performance results.

Chen et al. [17] have considered the comparison of algorithms as a stochastic optimization problem. The authors use stochastic optimization approaches to solve the algorithm selection problem. The scheme is useful for parameter tuning of algorithms on different sets of problem instances. But the comparison is expensive computationally and can be relaxed by assumption of exponential convergence property. This reduces the computation, however the assumption is not valid. The downsides of this approach is that it is complicated to implement and hard to interpret and it is not able to present a general sense about the algorithm performance.

A visual tool is developed by Dolan and Mor [35] and is called performance profile. The performance profile provides a plot including the probability of success rate for each algorithm on different instances of a benchmark. In other words, the percentage of problems which are solved within a factor of the best solve time is plotted. The advantages of these plots are showing visually how an algorithm runs over time and the interpretation contains both speed and success rate. But the main drawback is that it is not possible to compare how algorithms are close to the optimal and there are only two states of "optimal solution is found" and "optimal solution is not found yet" and the quality of solutions are not considered in this metric. Another visual comparison method is proposed by Ribeiro et al. [83]. The authors develop a framework for the run time distributions or time-to-target plots which is

applied and extended in [82]. The time-to-target plot is a 2-dimensional plot which y-axis is the probability that an algorithm will find a solution at least as good as a given target value within a given running time which is shown on x-axis. The target values are choosen as the objective values found by the state-of-the-art algorithms. This visualization is a useful tool for comparing algorithms as it provides an overview to the algorithm performance alongside with a statistical estimates of the errors. However, the main issues are that the choice of the target value is arbitrary and may skew the comparison results. Moreover, the uncertainty of the comparison is not identified in this approach and the superiority of the studied algorithm over the other is not presented clearly.

Considering the above drawbacks of different comparison approaches, we design a framework which provides an extensive overview to the algorithm performance along with a strong statistical conclusions about the significance of the observed differences. In a comprehensive study, Wolpert and Macready [94] have explored the relation of optimization algorithms and problems theoretically. The authors see the algorithm performance from the probability theory point of view. They propose the probability of getting better solution by one algorithm on a specific problem instance as a measure of algorithm performance. Their proposed framework establishes a theoretical and hypothetical approach to compare algorithms and the idea of our paper is similar to this scheme in a practical presence. Our approach is constructed based on probability dominance notions in decision theory and is augmented by statistical models and inferences.

4.3 Notations and Definitions

4.3.1 Probability Dominance

There exist arguments in decision making under uncertainty which are simple and effective, such as "A is likely to outperform B, so A is chosen". The probability dominance concept is introduced by Wrather and Yu [95] to provide a method to compare random outcomes. Assume that the smaller values of random outcome payoffs a better earning. Consider X and

Y are two random outcomes, according to the definition, X dominates Y with probability $\beta \geq 0.5$ and is shown by $X\beta Y$ if and only if:

$$P(X < Y) \geq \beta$$

where $P(X < Y)$ is the probability of that X outperform Y, and β is the likelihood of outperformance. It is defined that $\beta \geq 0.5$ and by this value for β, we can conclude from $X\beta Y$ that X is likely to outperform Y and Y is not likely to outperform X. Likely means something more than fifty-fifty chance of occurrence. The authors prove theorems about some properties of the concept and one of them is that probability dominance is not a transitive relation in general. According to this property, the definition of probability dominance is required to suitably modified based on the studying problem and the value of β may be changed to a proper value.

In order to apply this idea on algorithm comparison, consider A and B are two random outcomes of performance measure (e.g., objective value) obtained by algorithms $\mathcal{A}$ and $\mathcal{B}$ respectively. We have three possible cases in which $A < B$, $A = B$, and $A > B$. Without loss of generality, assume the problems are minimization. We are required to define the value of β based on our preferences of payoffs. For example, in the case of that someone is interested in *absolute superiority* of an algorithm, the equality of outcomes ($A = B$) is not considered and $\beta = 0.5$ provides outperformance. In another example, one may concern about the *highest assurance* payoffs of an algorithm compared to another. So, all three possible cases are considered and it is useful to allow β to be less than 0.5 and set $\beta = \frac{1}{3} \approx 0.34$. This problem is similar to the gambler's problem in [95].

4.3.2 Notations

For a pairwise comparison, we consider two algorithms $\mathcal{A}$ and $\mathcal{B}$ on a set of benchmark instances Ω. To estimate the related parameters, we repeatedly run $\mathcal{A}$ and $\mathcal{B}$ on each instance in Ω and record corresponding performance measures for each run (e.g., time to optimality, and best objective value). Let $X_\omega^{\mathcal{A}}$ denote a vector of performance measures (e.g., objective function value) obtained by repeatedly executing $\mathcal{A}$ on a problem $\omega \in \Omega$.

The notations are clarified in Table 4.1, where n_i correspond to the total number of runs for $\mathcal{A}$ on ω_i, and similarly m_i denotes number of runs for $\mathcal{B}$ on ω_i. Assume that the selected benchmark Ω includes k problems $\{\omega_1, \omega_2, \ldots, \omega_k\}$.

Table 4.1: Notations of the proposed framework.

Problem	Algorithm $\mathcal{A}$	Algorithm $\mathcal{B}$
ω_1	$X^{\mathcal{A}}_{\omega_1} = \{x^1_1, x^1_2, \ldots, x^1_{n_1}\}$	$X^{\mathcal{B}}_{\omega_1} = \{y^1_1, y^1_2, \ldots, y^1_{m_1}\}$
ω_2	$X^{\mathcal{A}}_{\omega_2} = \{x^2_1, x^2_2, \ldots, x^2_{n_2}\}$	$X^{\mathcal{B}}_{\omega_2} = \{y^2_1, y^2_2, \ldots, y^2_{m_2}\}$
$\ldots$	$\ldots$	$\ldots$
ω_k	$X^{\mathcal{A}}_{\omega_k} = \{x^k_1, x^k_2, \ldots, x^k_{n_k}\}$	$X^{\mathcal{B}}_{\omega_k} = \{y^k_1, y^k_2, \ldots, y^k_{m_k}\}$

At each iteration or time period, the data similar to Table 4.1 can be obtained from algorithms. By the following statistical frameworks, it is possible to calculate probability of getting less, equal, or more performance measures by one algorithm compared to the other one. To be more specific, consider $X^{\mathcal{A}}$ and $X^{\mathcal{B}}$ are two random outcomes from implementing algorithms $\mathcal{A}$ and $\mathcal{B}$ on problems Ω. The proposed framework provides the following probabilities:

$$P(X^{\mathcal{A}} < X^{\mathcal{B}}) \text{ which is denoted by } P_{\mathcal{A}<\mathcal{B}|\Omega}$$
$$P(X^{\mathcal{A}} = X^{\mathcal{B}}) \text{ which is denoted by } P_{\mathcal{A}=\mathcal{B}|\Omega}$$
$$P(X^{\mathcal{A}} > X^{\mathcal{B}}) \text{ which is denoted by } P_{\mathcal{A}>\mathcal{B}|\Omega}$$

The models also provide intervals in which the parameters lie with a specific level, i.e., confidence interval contains the true value. The parametric statistical model uses binomial in Section 4.4 and the non-parametric model is based on bootstrapping in Section 4.5. By using these probability and confidence intervals, it is possible to establish a comparison of algorithm performance based on probability dominance.

4.4 Binomial Models

In this section we assume that the number of evaluations is equal for algorithm $\mathcal{A}$ and $\mathcal{B}$ ($n_i = m_i$ for $i = 1, \ldots, k$), which can be easily achieved by downsampling. In other words, we can generate pairs of samples (x, y)'s of size $\mathcal{N}_\omega = \min\{|X_\omega^{\mathcal{A}}|, |X_\omega^{\mathcal{B}}|\}$. Each set of $X_\omega^{\mathcal{A}}$ and $X_\omega^{\mathcal{B}}$ is randomly downsized to $\mathcal{N}_\omega$ elements. The downsized sample $D_\omega^{\mathcal{A},\mathcal{B}}$ includes pairs of (x, y) from downsized sets of $X_\omega^{\mathcal{A}}$ and $X_\omega^{\mathcal{B}}$ which are still denoted as $X_\omega^{\mathcal{A}}$ and $X_\omega^{\mathcal{B}}$.

$$D_\omega^{\mathcal{A},\mathcal{B}} = \{(x_1, y_1), (x_2, y_2), ..., (x_{\mathcal{N}_\omega}, y_{\mathcal{N}_\omega})\}, x_i \in X_\omega^{\mathcal{A}} \text{ and } y_i \in X_\omega^{\mathcal{B}} \tag{4.1}$$

In the following discussion, we calculate the estimation for the probabilities with confidence intervals to provide assessment of the errors. The indicator functions can be defined as follows.

$$\begin{aligned}
1_{x<y}(x, y) &= \begin{cases} 1, & \text{if } x < y \\ 0, & \text{otherwise.} \end{cases} \\
1_{x>y}(x, y) &= \begin{cases} 1, & \text{if } x > y \\ 0, & \text{otherwise.} \end{cases} \\
1_{x=y}(x, y) &= \begin{cases} 1, & \text{if } x = y \\ 0, & \text{otherwise.} \end{cases}
\end{aligned} \tag{4.2}$$

Using the indicator functions in (4.2), we can count the number of performance metric pairs, where the performance metric values of $\mathcal{A}$ are less, equal or greater than the performance metrics of $\mathcal{B}$ on a particular problem ω.

$$N_{\mathcal{A}<\mathcal{B}|\omega} = \sum_{(x,y)\in D_\omega^{\mathcal{A},\mathcal{B}}} 1_{x<y} \tag{4.3}$$

$$N_{\mathcal{A}=\mathcal{B}|\omega} = \sum_{(x,y)\in D_\omega^{\mathcal{A},\mathcal{B}}} 1_{x=y} \tag{4.4}$$

$$N_{\mathcal{A}>\mathcal{B}|\omega} = \sum_{(x,y)\in D_\omega^{\mathcal{A},\mathcal{B}}} 1_{x>y} \tag{4.5}$$

On a random instance from Ω, the total number of runs which are better, same, and worse are respectively an aggregation of $N_{\mathcal{A}<\mathcal{B}|\omega}$, $N_{\mathcal{A}=\mathcal{B}|\omega}$ and $N_{\mathcal{A}>\mathcal{B}|\omega}$ over all problems in Ω and are calculated as

$$N_{\mathcal{A}<\mathcal{B}|\Omega} = \sum_{\omega \in \Omega} N_{\mathcal{A}<\mathcal{B}|\omega} \tag{4.6}$$

$$N_{\mathcal{A}=\mathcal{B}|\Omega} = \sum_{\omega \in \Omega} N_{\mathcal{A}=\mathcal{B}|\omega} \tag{4.7}$$

$$N_{\mathcal{A}>\mathcal{B}|\Omega} = \sum_{\omega \in \Omega} N_{\mathcal{A}>\mathcal{B}|\omega} \tag{4.8}$$

4.4.1 Parameters Estimation

Now we can estimate the probability of $\mathcal{A}$ having less, equal, or greater performance metric value than $\mathcal{B}$ on a problem $\omega \in \Omega$. To avoid replication formulation for calculations, let $\bowtie$ is a relational operator which is one of $\{<, =, >\}$ in the rest of this chapter.

$$P_{\mathcal{A}\triangleright\triangleleft\mathcal{B}|\omega} = \frac{1}{\mathcal{N}_\omega} \sum_{(x,y) \in D_\omega^{\mathcal{A},\mathcal{B}}} 1_{x \triangleright\triangleleft y}(x, y) \tag{4.9}$$

or

$$P_{\mathcal{A}\triangleright\triangleleft\mathcal{B}|\omega} = \frac{N_{\mathcal{A}\triangleright\triangleleft\mathcal{B}|\omega}}{\mathcal{N}_\omega} \tag{4.10}$$

Assume each instance ω is equally likely to be selected from Ω. The probability that $\mathcal{A}$ produces less, equal, or greater performance metric than $\mathcal{B}$ on a random instance from Ω is an average of $P_{\mathcal{A}\triangleright\triangleleft\mathcal{B}|\omega}$ across all problems in Ω.

$$P_{\mathcal{A}\bowtie\mathcal{B}|\Omega} = \frac{1}{|\Omega|} \sum_{\omega \in \Omega} P_{\mathcal{A}\triangleright\triangleleft\mathcal{B}|\omega} \tag{4.11}$$

or

$$P_{\mathcal{A}\bowtie\mathcal{B}|\Omega} = \frac{N_{\mathcal{A}\triangleright\triangleleft\mathcal{B}|\Omega}}{\mathcal{N}_\Omega} \tag{4.12}$$

where $|\Omega|$ is the number of instances in Ω, and $\mathcal{N}_\Omega$ is the total number of runs that were used to produce the metrics and equals to the following.

$$\mathcal{N}_\Omega = \sum_{\omega \in \Omega} \mathcal{N}_\omega = N_{\mathcal{A}<\mathcal{B}|\Omega} + N_{\mathcal{A}=\mathcal{B}|\Omega} + N_{\mathcal{A}>\mathcal{B}|\Omega} \tag{4.13}$$

An estimate of the probability does not provide any information about its accuracy. In order to provide such estimate, we calculate the confidence interval for the estimated value with a nominal confidence level $(1 - \alpha\%)$. Since the probabilities are binomial proportions and discrete, it is not possible to calculate the exact nominal confidence level. A confidence interval is preferred when the actual coverage probability is close to the nominal confidence level. In the following discussion, we present different types of interval for binomial proportions along with their advantages and disadvantages.

The **Wald** Interval approximation is defined based on normal theory approximation. The upper and lower bound of interval with the nominal confidence level of $1 - \alpha$ for $P_{\mathcal{A}\rhd\lhd\mathcal{B}|\Omega}$ is defined in (4.14). This type is easy to calculate and popular in practice but it has a poor performance when the sample size is small which is studied in [3] and the actual coverage probability is also poor when the point of interest is near to zero or one according to [13].

$$\begin{aligned} L_{CI} &= P_{\mathcal{A}\rhd\lhd\mathcal{B}|\Omega} - z_{\frac{\alpha}{2}} \sqrt{\frac{P_{\mathcal{A}\rhd\lhd\mathcal{B}|\Omega}(1 - P_{\mathcal{A}\rhd\lhd\mathcal{B}|\Omega})}{\mathcal{N}_\Omega}} \\ U_{CI} &= P_{\mathcal{A}\rhd\lhd\mathcal{B}|\Omega} + z_{\frac{\alpha}{2}} \sqrt{\frac{P_{\mathcal{A}\rhd\lhd\mathcal{B}|\Omega}(1 - P_{\mathcal{A}\rhd\lhd\mathcal{B}|\Omega})}{\mathcal{N}_\Omega}} \end{aligned} \tag{4.14}$$

where $z_{\frac{\alpha}{2}}$ is the $1 - z_{\frac{\alpha}{2}}$ quantile of the standard normal distribution and $\mathcal{N}_\Omega$ is the total number of runs which is discussed before.

An alternative approach constructs confidence interval based on reverting the one-tailed hypothesis test procedure for the null hypothesis $H_0 : p = p_0$ as in (4.15). Since the interval estimator is constructed to have at least $(1-\alpha)\%$ coverage probability for every proportion, this approach is called exact method [19].

$$\sum_{j=x}^{n} \binom{n}{j} p_0^j (1-p_0)^{n-j} = \frac{\alpha}{2}$$
$$\sum_{j=0}^{x} \binom{n}{j} p_0^j (1-p_0)^{n-j} = \frac{\alpha}{2} \tag{4.15}$$

This approach is also known as **Clopper-Pearson** confidence interval and the lower and upper bound of the interval with the nominal confidence level $1-\alpha$ for $P_{\mathcal{A}\bowtie\mathcal{B}|\Omega}$ is defined in (4.16). This type of interval guarantees the coverage probability and is applied to avoid interval approximation but it is too conservative which is investigated in [3]. In other words, the actual coverage probability is much larger than the nominal confidence level. This difference between actual and nominal level can be negligible for a quite large sample size.

$$L_{CI} = \frac{N_{\mathcal{A}\bowtie\mathcal{B}|\Omega}}{N_{\mathcal{A}\bowtie\mathcal{B}|\Omega} + (\mathcal{N}_\Omega - N_{\mathcal{A}\bowtie\mathcal{B}|\Omega} + 1) F^{\nu_1}_{\nu_2,(1-\frac{\alpha}{2})}}$$
$$U_{CI} = \frac{(N_{\mathcal{A}\bowtie\mathcal{B}|\Omega} + 1) F^{\nu_3}_{\nu_4,\frac{\alpha}{2}}}{\mathcal{N}_\Omega - N_{\mathcal{A}\bowtie\mathcal{B}|\Omega} + (N_{\mathcal{A}\bowtie\mathcal{B}|\Omega} + 1) F^{\nu_3}_{\nu_4,\frac{\alpha}{2}}} \tag{4.16}$$

where $F^d_{e,q}$ represents the q quantile from an F-distribution with d and e degrees of freedom in which $\nu_1 = 2N_{\mathcal{A}\bowtie\mathcal{B}|\Omega}$, $\nu_2 = 2(\mathcal{N}_\Omega - N_{\mathcal{A}\bowtie\mathcal{B}|\Omega} + 1)$, $\nu_3 = 2(N_{\mathcal{A}\bowtie\mathcal{B}|\Omega} + 1)$, and $\nu_4 = 2(\mathcal{N}_\Omega - N_{\mathcal{A}\bowtie\mathcal{B}|\Omega})$.

There exists another method which is the inverse of the Wald method procedure by considering null hypothesis $H_0 : P_{\mathcal{A}\bowtie\mathcal{B}|\Omega} = p_0$ on the approximate normal test. In other words, the lower and upper bound are calculated by solving the equation (4.17). The approach is first discussed by [93] and is known as **Wilson's score** interval.

$$\frac{P_{\mathcal{A}\bowtie\mathcal{B}|\Omega} - p_0}{\sqrt{\frac{p_0(1-p_0)}{\mathcal{N}_\Omega}}} = \pm z_{\frac{\alpha}{2}} \tag{4.17}$$

The Wilson's score interval has a coverage probability close to nominal confidence level as discussed in [2]. When comparing with the Wald interval and Clopper-Pearson intervals, the Wilson's score performs better for any sample sizes and parameter values according to [3]. On the other hand, Wilson's score method has a poor coverage probability near 0 or 1 which is below the nominal confidence level. The lower and upper bound of the Wilson's score interval with the nominal confidence level of $(1-\alpha)\%$ for $P_{\mathcal{A}\bowtie\mathcal{B}|\Omega}$ is formulated as follows:

$$L_{CI} = \frac{P_{\mathcal{A}\bowtie\mathcal{B}|\Omega} + \frac{z^2_{\frac{\alpha}{2}}}{2\mathcal{N}_\Omega} - z_{\frac{\alpha}{2}}\sqrt{\frac{P_{\mathcal{A}\bowtie\mathcal{B}|\Omega}(1-P_{\mathcal{A}\bowtie\mathcal{B}|\Omega})+\frac{z^2_{\frac{\alpha}{2}}}{4\mathcal{N}_\Omega}}{\mathcal{N}_\Omega}}}{1+\frac{z^2_{\frac{\alpha}{2}}}{\mathcal{N}_\Omega}}$$

$$U_{CI} = \frac{P_{\mathcal{A}\bowtie\mathcal{B}|\Omega} + \frac{z^2_{\frac{\alpha}{2}}}{2\mathcal{N}_\Omega} + z_{\frac{\alpha}{2}}\sqrt{\frac{P_{\mathcal{A}\bowtie\mathcal{B}|\Omega}(1-P_{\mathcal{A}\bowtie\mathcal{B}|\Omega})+\frac{z^2_{\frac{\alpha}{2}}}{4\mathcal{N}_\Omega}}{\mathcal{N}_\Omega}}}{1+\frac{z^2_{\frac{\alpha}{2}}}{\mathcal{N}_\Omega}} \tag{4.18}$$

where $z_{\frac{\alpha}{2}}$ is the $1-z_{\frac{\alpha}{2}}$ quantile of the standard normal distribution.

Since the Wilson's score formula (4.18) is difficult to interpret, a modification was applied to the simplest approach (Wald interval) by [3] which is called **adjusted Wald** interval. In order to construct 95% confidence interval, we have $z^2_{\frac{\alpha}{2}} = 1.96^2 \approx 4$ which the Wilson's score formulation becomes similar to ordinary Wald interval where we add two successes and two fails to the number of runs. This simple modification changes the interval from highly liberal to slightly conservative. It is a little more conservative than Wilson's score, especially for small size samples according to [13]. This method is recommended when the sample size is larger than 40 ($n \geq 40$). It is easy to formulate this approach as described and the lower and upper bound of adjusted Wald interval with the nominal confidence interval of $1-\alpha$ for $P_{\mathcal{A}\bowtie\mathcal{B}|\Omega}$ is shown in the followings.

$$L_{CI} = P'_{\mathcal{A}\bowtie\mathcal{B}|\Omega} - z_{\frac{\alpha}{2}}\sqrt{\frac{P'_{\mathcal{A}\bowtie\mathcal{B}|\Omega}(1-P'_{\mathcal{A}\bowtie\mathcal{B}|\Omega})}{\mathcal{N}_\Omega+4}}$$
$$U_{CI} = P'_{\mathcal{A}\bowtie\mathcal{B}|\Omega} + z_{\frac{\alpha}{2}}\sqrt{\frac{P'_{\mathcal{A}\bowtie\mathcal{B}|\Omega}(1-P'_{\mathcal{A}\bowtie\mathcal{B}|\Omega})}{\mathcal{N}_\Omega+4}} \tag{4.19}$$

where $z_{\frac{\alpha}{2}}$ is the $1 - z_{\frac{\alpha}{2}}$ quantile of the standard normal distribution and $P'_{\mathcal{A}\bowtie\mathcal{B}|\Omega} = \frac{N_{\mathcal{A}\bowtie\mathcal{B}|\Omega}+2}{\mathcal{N}_\Omega+4}$.

4.5 Bootstrap

In order to avoid the assumptions of the previous techniques and downsampling on the dataset, we introduce a strong statistical technique in this section in which the confidence intervals are more reliable than other techniques. Bootstrap is an empirical statistical technique which is popularized by [37] and is simple to implement but is more accurate due to computations on data itself to estimate the variation. In precise description, the technique repeatedly samples from the dataset to estimate the variation.

The first step of this method is *re-sampling* to estimate the probabilities. Consider runs which are presented in Table 4.1. First, we random sample n_1 runs with replacement from $X^{\mathcal{A}}_{\omega_1}$ and random sample m_1 runs with replacement from $X^{\mathcal{B}}_{\omega_1}$. We denote the new samples by $X'^{\mathcal{A}}_{\omega_1} = (x'^1_1, x'^1_2, \ldots, x'^1_{n_1})$ and $X'^{\mathcal{B}}_{\omega_1} = (y'^1_1, y'^1_2, \ldots, y'^1_{m_1})$ respectively. Second, we generate the pairs set from $X'^{\mathcal{A}}_{\omega_1}$ and $X'^{\mathcal{B}}_{\omega_1}$ by Cartesian product-wise. So, we have all pairs from two sets as is shown in the followings.

$$\begin{aligned} O^{\mathcal{A},\mathcal{B}}_{\omega} = \{&(x'_1, y'_1), (x'_1, y'_2), \ldots, (x'_1, y'_{m_1}),\\ &(x'_2, y'_1), (x'_2, y'_2), \ldots, (x'_2, y'_{m_1}),\\ &\ldots\\ &(x'_{n_1}, y'_1), (x'_{n_1}, y'_2), \ldots, (x'_{n_1}, y'_{m_1})\},\\ &x'_i \in X^{\mathcal{A}}_{\omega} \text{ and } y'_i \in X^{\mathcal{B}}_{\omega} \end{aligned} \tag{4.20}$$

Next, in calculations similar to (4.3), (4.4), and (4.5), we calculate the comparisons as follows:

$$N^r_{\mathcal{A}<\mathcal{B}|\omega} = \sum_{(x,y)\in O^{\mathcal{A},\mathcal{B}}_{\omega}} 1_{x<y} \tag{4.21}$$

$$N^r_{\mathcal{A}=\mathcal{B}|\omega} = \sum_{(x,y)\in O^{\mathcal{A},\mathcal{B}}_{\omega}} 1_{x=y} \tag{4.22}$$

$$N^r_{\mathcal{A}>\mathcal{B}|\omega} = \sum_{(x,y)\in O^{\mathcal{A},\mathcal{B}}_{\omega}} 1_{x>y} \tag{4.23}$$

where $1_{x<y}(x,y)$, $1_{x>y}(x,y)$, and $1_{x=y}(x,y)$ are indicator functions as in (4.2). Moreover, the aggregations of different problems are calculated as follows.

$$N^r_{\mathcal{A}<\mathcal{B}|\Omega} = \sum_{\omega\in\Omega} N^r_{\mathcal{A}<\mathcal{B}|\omega} \tag{4.24}$$

$$N^r_{\mathcal{A}=\mathcal{B}|\Omega} = \sum_{\omega\in\Omega} N^r_{\mathcal{A}=\mathcal{B}|\omega} \tag{4.25}$$

$$N^r_{\mathcal{A}>\mathcal{B}|\Omega} = \sum_{\omega\in\Omega} N^r_{\mathcal{A}>\mathcal{B}|\omega} \tag{4.26}$$

The probabilities are calculated as follows.

$$P^r_{\mathcal{A}\bowtie\mathcal{B}|\Omega} = \frac{N^r_{\mathcal{A}\bowtie\mathcal{B}|\Omega}}{\mathcal{N}\times\mathcal{M}} \tag{4.27}$$

where $\mathcal{N}\times\mathcal{M} = \sum_{i=1}^{k} n_i \times m_i$ and $\bowtie$ is is a relational operator of $\{<,=,>\}$.

We repeat the process from the re-sampling step for R replicates independently and record values from (4.27) for each iteration r. Then, we report the average of these values as an approximation for the probabilities which are calculated as follows.

$$P_{\mathcal{A}\bowtie\mathcal{B}|\Omega} = \frac{\sum_{r=1}^{R} P^r_{\mathcal{A}\bowtie\mathcal{B}|\Omega}}{R} \tag{4.28}$$

Similar to binomial methods, an estimate of the measures does not provide any information about its accuracy and we need to calculate the confidence interval which is the estimated value with a nominal confidence level $(1-\alpha)\%$. There exist different methods for calculating the confidence interval for bootstrapping approach according to [30]. A generic confidence interval is adjusted for **normal** estimation which the bootstrap bias is considered and calculated as follows.

$$L_{CI} = \hat{\theta} - z_{\frac{\alpha}{2}} \times SE(\hat{\theta})$$
$$U_{CI} = \hat{\theta} + z_{\frac{\alpha}{2}} \times SE(\hat{\theta})$$

where $\hat{\theta}$ is the estimation of parameter of interest ($P_{\mathcal{A}\triangleright\triangleleft\mathcal{B}|\Omega}$) and $z_{\frac{\alpha}{2}}$ is the $1 - z_{\frac{\alpha}{2}}$ quantile of the standard normal distribution. Moreover, $SE(\hat{\theta})$ is the estimation of standard error which is calculated in the following.

$$SE(\hat{\theta}) = \Big[\sum_{r=1}^{R} \frac{(\hat{\theta_r} - \hat{\theta})^2}{R-1}\Big]^{\frac{1}{2}} \tag{4.29}$$

where $\hat{\theta_r}$ is the statistic of interest in each replication r ($P^r_{\mathcal{A}\triangleright\triangleleft\mathcal{B}|\Omega}$). The normal approximation method requires large size sample and shows poor performance on coverage probability. The using bootstrap is especially useful when the distribution of the parameters are unknown. The **percentile** method uses the bootstrap distribution itself. The lower-bound and upper-bound of the interval are the quantiles of parameters in all bootstrap replications. In other words, we sort the values of (4.27) (e.g., $\{\hat{\theta}^*_1, \hat{\theta}^*_2, \ldots, \hat{\theta}^*_R\}$). The lower-bound is the interpolation of $\frac{\alpha}{2} \times R$-th of the vectors and the interpolation of $(1 - \frac{\alpha}{2}) \times R$-th of the vectors is considered as the upper-bound as follows.

$$L_{CI} = \hat{\theta}^*_{\frac{\alpha}{2}\times R}$$
$$U_{CI} = \hat{\theta}^*_{(1-\frac{\alpha}{2})\times R}$$

where $\hat{\theta}^*_i$'s are the sorted vector of parameter of interest for all replications. Although the percentile method is highly symmetric and simple to calculate, there exist substantial

coverage error and tends to be narrow for small size samples. Another method for confidence interval is **basic bootstrap** which tries to estimate the distribution of parameter by the empirical replications. The lower-bound and upper bound of the method are in the followings.

$$L_{CI} = 2\hat{\theta} - \hat{\theta}^*_{(1-\frac{\alpha}{2})\times R}$$
$$U_{CI} = 2\hat{\theta} - \hat{\theta}^*_{\frac{\alpha}{2}\times R}$$

where $\hat{\theta}$ is the estimation of parameter of interest ($P_{\mathcal{A}\bowtie\mathcal{B}|\Omega}$) and $\hat{\theta}^*_i$'s are the sorted vector of parameter of interest for all replications. This method is simple and reasonably accurate, but it does not have correctness for bias and skewness. Bias-corrected and accelerated (**BCa**) method is proposed by Efron [38] to correct the bias and skew. The lower-bound and upper-bound of the confidence interval are calculated in the followings.

$$L_{CI} = \hat{\theta}^*_{\alpha_1\times R}$$
$$U_{CI} = \hat{\theta}^*_{\alpha_2\times R}$$

where $\hat{\theta}^*_i$'s are the sorted vector of parameter of interest for all replications and α_1 and α_2 are computed as follows.

$$\alpha_1 = \Phi\Big(\hat{z} + \frac{\hat{z} + z_{\frac{\alpha}{2}}}{1 - \hat{a}.(\hat{z} + z_{\frac{\alpha}{2}})}\Big)$$
$$\alpha_2 = \Phi\Big(\hat{z} + \frac{\hat{z} + z_{1-\frac{\alpha}{2}}}{1 - \hat{a}.(\hat{z} + z_{1-\frac{\alpha}{2}})}\Big)$$

where Φ is cumulative distribution function (CDF) of the standard normal distribution, $\hat{z}$ is bias parameter, and $\hat{a}$ is skewness correction factor. The values of $\hat{z}$ and $\hat{a}$ are calculated in (4.30) and (4.31) respectively.

$$\hat{z} = \Phi^{-1}\big(\frac{\sum_{r=1}^{R} 1_{\hat{\theta}_r<\hat{\theta}}(\hat{\theta}_r, \hat{\theta})}{R}\big) \tag{4.30}$$

where $1_{\hat{\theta_r}<\hat{\theta}}(\hat{\theta_r}, \hat{\theta})$ is the indicator functions, $\hat{\theta_r}$ is the statistic of interest in each replication r ($P^r_{\mathcal{A}\rhd\lhd\mathcal{B}|\Omega}$), and $\hat{\theta}$ is the estimation of parameter of interest ($P_{\mathcal{A}\rhd\lhd\mathcal{B}|\Omega}$). In other words, $\hat{z}$ equals to the inverse CDF of probability of that $\hat{\theta_r} < \hat{\theta}$ on all R replications.

$$\hat{a} = \frac{\sum_{i=1}^{n}(\hat{\theta}_{(.)} - \hat{\theta}_{(-i)})^3}{6\big[\sum_{i=1}^{n}(\hat{\theta}_{(.)} - \hat{\theta}_{(-i)})^2\big]^{\frac{3}{2}}} \tag{4.31}$$

where $\hat{\theta}_{(-i)}$ is the estimation of parameter of interest when the i-th sample is eliminated from the original dataset. This value is $P_{\mathcal{A}\rhd\lhd\mathcal{B}|\Omega}$ while the i-th sample is removed from the original dataset. In the equation (4.31), the dataset size is denoted by n, and $\hat{\theta}_{(.)} = \sum_{i=1}^{n}\frac{\hat{\theta}_{(-i)}}{n}$. Although the **BCa** method is rather complicated and computationally expensive, the confidence interval is more accurate than other bootstrapping because it is similar to percentile with considering that is un-biased and skewness corrected.

When we are using bootstrap method, it is required to keep in mind that re-sampling is not able to improve the parameter of interest estimate. Even a fair amount of trials, there will be differences between empirical distribution and the true distribution. But the variation is less sensitive to differences, and empirical and true distributions will show similar variations if they are fairly close. Therefore, the bootstrap method is more robust than binomial approaches.

The bootstrap method is based on replications of re-sampling (R). Most of the literature recommends to choose a large number of replicates to reduce the estimation error. But it can be computationally expensive in the case of large size dataset sample. Although the method highly depends on the dataset, the question is how many replicates (R) is enough?

The confidence interval methods imply that to choose R so that $\frac{\alpha}{2} \times (R+1)$ is an integer which for nominal confidence interval level 95% ($\alpha = 0.05$) and 99% ($\alpha = 0.01$), the smallest values for R are 39 and 199 respectively. Choosing these smallest values can result in power loss of bootstrap test. Efron and Tibshirani [39] suggest that choosing $R = 500$ is adequate for most cases in general. Manly [70] advises to set $R = 200$ in order to have a fairly small error. Moreover, Wilcox [91] recommends $R = 599$ for general application and Davidson and MacKinnon [25] provide number of replications with quite small error for different confidence

interval levels which changes from $R = 399$ to $R = 1499$ for nominal levels of 95% to 99%. In spite of these studies, one may use several thousands replicates in a conservative standpoint.

4.6 Applications

The probability of getting better solution can be calculated by the statistical models and can be interpreted through probability dominance concept based on desired payoffs. The probabilities are calculated for a specific run time of algorithms which are referred to data at a specific time t similar to Table 4.1. So, we can calculate different probabilities on set of instances of problems Ω for a period of time Δ and denote by $P^t_{\mathcal{A}\bowtie\mathcal{B}|\Omega}$ where t is time unit and $t = \{1, \ldots, \Delta\}$, and $\bowtie = \{>, =, <\}$. In order to see how the algorithm performance changes over time, we can plot the probabilities of each time unit in which the x-axis is time unit and y-axis is the probabilities. According to discussed payoffs in Section 4.3, we can plot the results in two ways. One plot includes probabilities of obtaining solutions by algorithm $\mathcal{A}$ less and greater than algorithm $\mathcal{B}$ ($P^t_{\mathcal{A}<\mathcal{B}|\Omega}$ and $P^t_{\mathcal{A}>\mathcal{B}|\Omega}$)) with their confidence intervals to see the trends of *absolute superiority* of algorithm performance which we call it *dominance plot*. Another way of showing the changes over a time period is to plot cumulative probabilities to see the trends of *highest assurance payoffs* that we call it *outcome plot*. Since the summation of all probabilities at each time is fixed and equals to 1 (i.e., $P^t_{\mathcal{A}<\mathcal{B}|\Omega} + P^t_{\mathcal{A}=\mathcal{B}|\Omega} + P^t_{\mathcal{A}>\mathcal{B}|\Omega} = 1$), more areas of plot shows more assurance of algorithm. We can see the plots in the followings in details.

In order to demonstrate the proposed methodology for comparing algorithms, assume that we have two algorithms $\mathcal{A}$ and $\mathcal{B}$. We generate random instances of performance measure (objective values) which the theoretical performance can be computed. To do so objective values are randomly generated by normal distribution with exponential decay over time for different parameters (test problems). The formulation of the test problems is as follows.

$$\text{objective value at time } t \sim \mathcal{N}\Big(\mu \times (1-\lambda)^t + \mu_c, \sigma^2\Big) \tag{4.32}$$

where λ is the decay rate for mean, μ_c is a fixed number which is added to the mean, and t is the time unit. The theoretical probabilities for each test problem can be determined as

follows. Assume each problem on two algorithms ($\mathcal{A}$ and $\mathcal{B}$) are generated independently by the above normal distribution with the following parameters.

$$\text{objective value is obtained by algorithm } \mathcal{A}\text{at time } t = X_{\mathcal{A}} \sim \mathcal{N}\Big(\mu_{\mathcal{A}}(t), \sigma^2_{\mathcal{A}}\Big)$$
$$\text{objective value is obtained by algorithm } \mathcal{B}\text{at time } t = X_{\mathcal{B}} \sim \mathcal{N}\Big(\mu_{\mathcal{B}}(t), \sigma^2_{\mathcal{B}}\Big)$$

where $\mu_{\mathcal{A}}(t)$ and $\mu_{\mathcal{B}}(t)$ are mean values which are functions of time unit (t). Therefore, the theoretical values for probability of obtaining solutions by algorithm $\mathcal{A}$ less than algorithm $\mathcal{B}$ is calculated in the followings.

$$\begin{aligned} P_{\mathcal{A}<\mathcal{B}|\Omega} &= P(X_{\mathcal{A}} < X_{\mathcal{B}}) \\ &= P(X_{\mathcal{A}} - X_{\mathcal{B}} < 0) \end{aligned}$$

where

$$X_{\mathcal{A}} - X_{\mathcal{B}} = Y \sim \mathcal{N}\Big(\mu_{\mathcal{A}}(t) - \mu_{\mathcal{B}}(t), \sigma^2_{\mathcal{A}} + \sigma^2_{\mathcal{B}}\Big) \tag{4.33}$$

Therefore, we have the following.

$$P_{\mathcal{A}<\mathcal{B}|\Omega} = P(X_{\mathcal{A}} - X_{\mathcal{B}} < 0) \tag{4.34}$$
$$= P(Y < 0) \tag{4.35}$$
$$= \frac{1}{\sqrt{(\sigma^2_{\mathcal{A}} + \sigma^2_{\mathcal{B}})2\pi}} \int_{-\infty}^{0} e^{-\frac{(Y-\mu_{\mathcal{A}}(t)+\mu_{\mathcal{B}}(t))^2}{2(\sigma^2_{\mathcal{A}}+\sigma^2_{\mathcal{B}})}} .dY \tag{4.36}$$

Similarly, the probability of getting solutions by algorithm $\mathcal{A}$ greater than algorithm $\mathcal{B}$ is as follows.

$$\begin{aligned} P_{\mathcal{A}>\mathcal{B}|\Omega} &= P(X_{\mathcal{A}} > X_{\mathcal{B}}) \\ &= P(X_{\mathcal{A}} - X_{\mathcal{B}} > 0) \end{aligned}$$

According to (4.33), we have the following.

$$P_{\mathcal{A}>\mathcal{B}|\Omega} = P(X_{\mathcal{A}} - X_{\mathcal{B}} > 0) \tag{4.37}$$

$$= P(Y > 0) \tag{4.38}$$

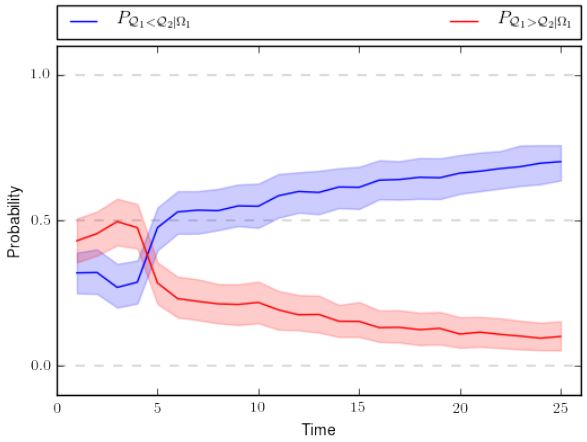
$$= \frac{1}{\sqrt{(\sigma_{\mathcal{A}}^2 + \sigma_{\mathcal{B}}^2)2\pi}} \int_0^{\infty} e^{-\frac{(Y-\mu_{\mathcal{A}}(t)+\mu_{\mathcal{B}}(t))^2}{2(\sigma_{\mathcal{A}}^2+\sigma_{\mathcal{B}}^2)}} .dY \tag{4.39}$$

In theoretical calculation, the probability of getting equal solutions by two algorithms is zero because in the random generation samples by a normal distribution the probability of equal solutions is:

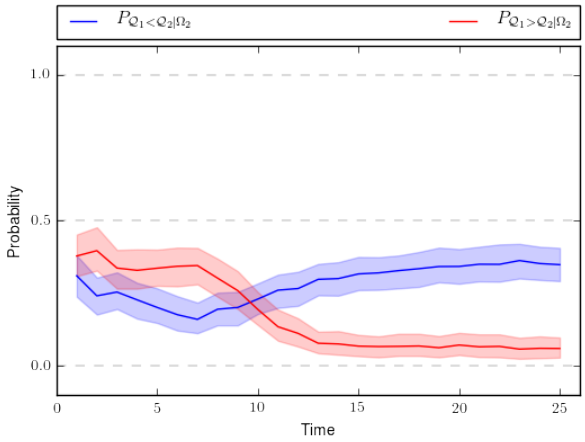
$$\begin{aligned} P(Y = 0) &= P(X_{\mathcal{A}} - X_{\mathcal{B}} = 0) \\ &= P(X_{\mathcal{A}} = X_{\mathcal{B}}) \\ &= 0 \end{aligned}$$

But in real situations, it is common that two different algorithms generate equal solutions for several runs which results in $P(Y = 0) > 0$. In such a condition, outcome plot helps the decision making process of selecting the better algorithm while dominance plot does not show the probability of equal solutions by two algorithms. In order to illustrate this, assume that a tabu search algorithm with two different set of parameters ($\mathcal{Q}_1$ and $\mathcal{Q}_2$) have been applied to two different sets of problems (Ω_1 and Ω_2). The dominance plots and outcome plots are shown in Figure 4.1.

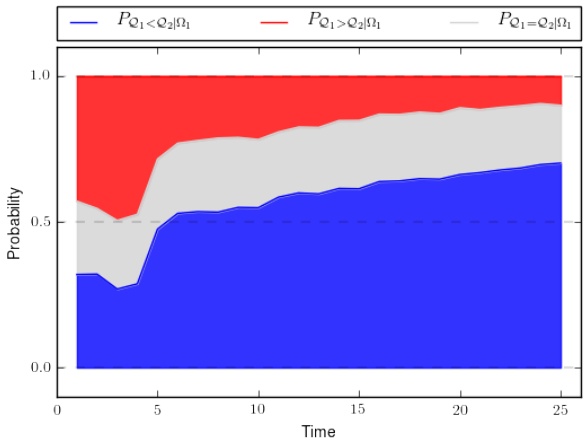
We can see that the dominance plot in Figure 4.1a is used to choose the better algorithm because the probability of getting better solutions by algorithm $\mathcal{Q}_1$ goes above the 0.5 after a while, even for the lower bound of the confidence interval. This shows the absolute superiority of algorithm $\mathcal{Q}_1$ over algorithm $\mathcal{Q}_2$ on the set of problems Ω_1. The outcome plot in Figure 4.1c is not required for making decision, however it provides a better understanding of algorithm performance. On the other hand, the probabilities remains under 0.5 in Figure 4.1b and we cannot use it to choose the better algorithm. By investigating the outcome plot in Figure 4.1d, it is seen that the probability of getting better solutions by algorithm $\mathcal{Q}_2$ goes to zero and the cumulative areas of probabilities of getting solutions by algorithm $\mathcal{Q}_1$ at least



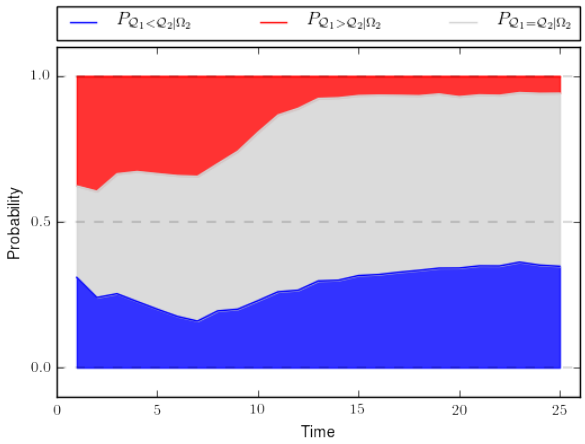
(a) dominance plot Ω_1



(b) dominance plot Ω_2



(c) outcome plot Ω_1



(d) outcome plot Ω_2

Figure 4.1: Plots of two tabu settings on two sets of problems with 95% confidence level.

as good as algorithm $\mathcal{Q}_2$ dominates the area under curve. So, the outcome plot helps the interpretation and revision of the payoffs preferences.

In order to see how the measures work on different examples, we have randomly generated a benchmark by the equation (4.32) including four test problems on 100 time units and 50 runs for each problem. The problems have different behaviors on algorithm $\mathcal{A}$ and algorithm $\mathcal{B}$ with different parameters values as is described in Table 4.2. The variance for all problems equal to 10^4 ($\sigma = 100$), and the average for both algorithms of each problem are set equally to make same starts for both algorithms.

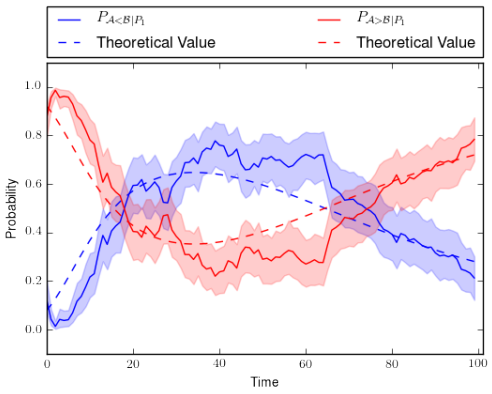
Table 4.2: Parameters of test problems generation.

Problem	Algorithm $\mathcal{A}$	Algorithm $\mathcal{B}$
Problem 1	$\mu = 1000, \lambda = 4\%, \mu_c = 1100$	$\mu = 1000, \lambda = 2\%, \mu_c = 900$
Problem 2	$\mu = 1500, \lambda = 1.01\%, \mu_c = 1700$	$\mu = 1500, \lambda = 1\%, \mu_c = 1700$
Problem 3	$\mu = 2000, \lambda = 1.20\%, \mu_c = 2200$	$\mu = 2000, \lambda = 1\%, \mu_c = 2200$
Problem 4	$\mu = 2300, \lambda = 1\%, \mu_c = 2500$	$\mu = 2300, \lambda = 1.10\%, \mu_c = 2500$

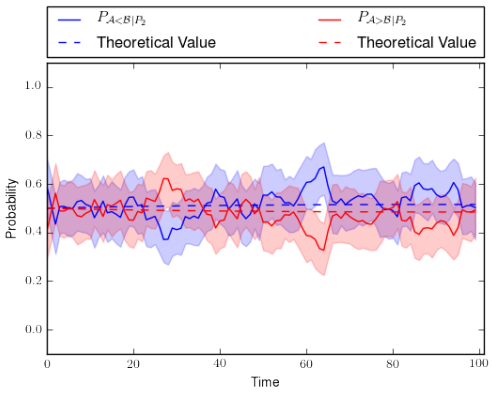
We tried to generate a benchmark that includes different possibilities for algorithms performance comparison and the values of parameters were regulated by trial and error. For example, problem 1 on algorithm $\mathcal{A}$ has a double rate of decrease in comparison to the algorithm $\mathcal{B}$, but the fixed value for the mean is larger. This is similar to a case in which an algorithm has a better rate for finding solutions than the counterpart algorithm, but it gets trapped in a local minimum after a while and its counterpart converges to a better solution. Another examples, problem 2 and problem 3 have similar generation method for both algorithms, except problem 2 is 1%, and problem 3 is 20% faster in decay rate for algorithm $\mathcal{A}$ than algorithm $\mathcal{B}$. Lastly, problem 4 has identical parameter values for both algorithms, aside from decay rate of algorithm $\mathcal{A}$ which is 10% slower than the decay rate of algorithm $\mathcal{B}$.

Since the probability of getting equal solutions is zero in theoretical random generation, we only display the dominance plot for each problem in the benchmark Figure 4.2. The plots include the values on the random instances and the theoretical values calculated in (4.36) and (4.39). Although the calculation of bootstrap method can be costly, it is more reliable and also smoother than binomial approaches. So, the confidence interval method for the

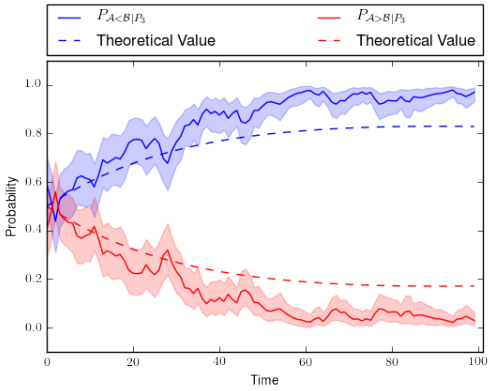
plots is BCa bootstrap which is known as the best accurate in the literature and the nominal level is 95% and the number of replicates in bootstrap is 1499 ($R = 1499$).



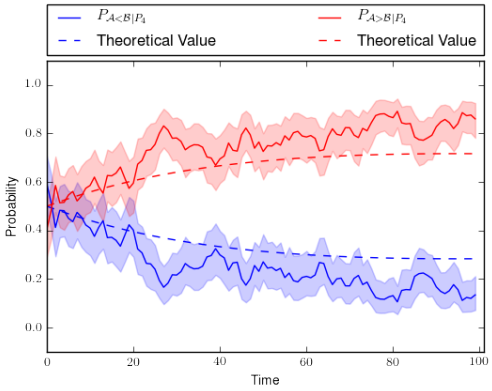
(a) Problem 1



(b) Problem 2



(c) Problem 3



(d) Problem 4

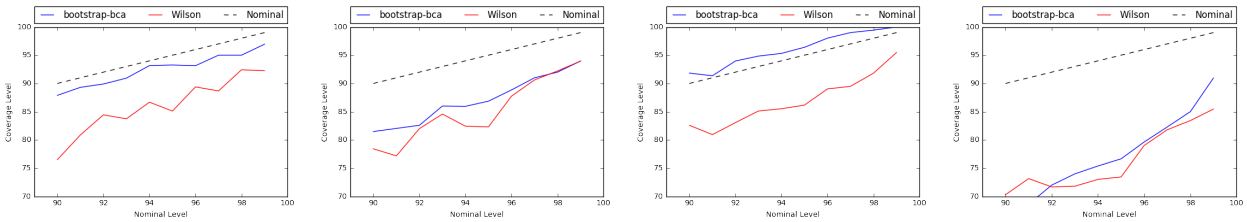
Figure 4.2: Dominance probability of each test problem with 95% confidence level and theoretical values.

In order to examine the coverage probability of binomial and bootstrap confidence interval for different nominal levels, we implemented a simulation as follows.

- We generated one random sample for all problems as is described in Table 4.2 and assumed this as the reference sample.
- The probabilities along with their confidence intervals were calculated for each problem.
- 100 new random samples were generated similar to the reference sample.
- The probabilities were calculated for all new samples.

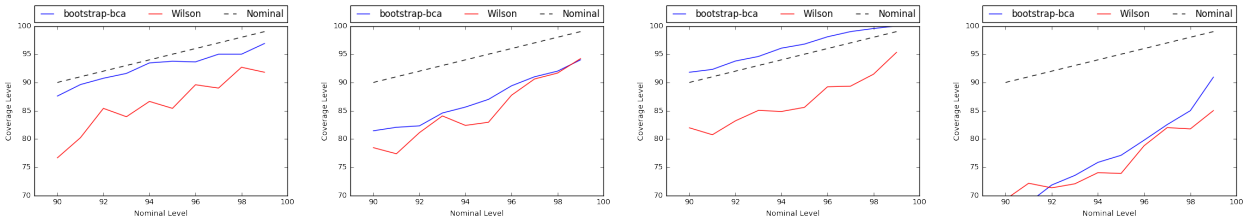
- The coverage probability is the probability of that new sample probabilities are covered by the reference sample confidence interval.

We investigated the simulation for Wilson's score and BCa methods in probability. The nominal confidence interval level changes from 90% to 99%. The coverage probability and nominal level for probability of getting solutions by algorithm $\mathcal{A}$ less than algorithm $\mathcal{B}$ ($P^t_{\mathcal{A}<\mathcal{B}|\Omega}$) of each problem are shown in Figure 4.3 and the coverage probability and nominal level for probability of getting solutions by algorithm $\mathcal{A}$ greater than algorithm $\mathcal{B}$ ($P^t_{\mathcal{A}>\mathcal{B}|\Omega}$) of each problem are shown in Figure 4.4. The x-axis and y-axis are the coverage percentage of different values in both Figures 4.3 and 4.4.



(a) Problem 1 **(b)** Problem 2 **(c)** Problem 3 **(d)** Problem 4

Figure 4.3: The coverage probability and nominal level for better solutions probabilities ($P^t_{\mathcal{A}<\mathcal{B}|\Omega}$).



(a) Problem 1 **(b)** Problem 2 **(c)** Problem 3 **(d)** Problem 4

Figure 4.4: The coverage probability and nominal level for worse solutions probabilities ($P^t_{\mathcal{A}>\mathcal{B}|\Omega}$).

We can see that bootstrap BCa confidence interval shows a good cover of nominal level over all probabilities. On the other hand, the Wilson's score method is anti-conservative for probabilities near to zero and one and has a poor performance in compare with bootstrap BCa.

4.7 Results

In order to demonstrate the proposed methodology for comparing algorithms, we present a comparison of the guided tabu algorithm (GTA) and the standard tabu algorithm. We have applied GTA algorithm which is described in 3.2.7, with the number of iterations equal to 300,000. The minimum and maximum of θ values equal to $\theta_{\min} = 0.0$ and $\theta_{\max} = 1.0$ according to the experiment have been done in 2.3. The standard tabu search algorithm have been also applied, with the number of iterations equal to 300,000. The number of epochs for both tabu and GTA is 200.

4.7.1 Test Problems

We applied algorithms to the Taillard's benchmark [88] and Demirkol's instances [26] of the job shop scheduling problem. The local search is adapted from [50] on both standard tabu and GTA. More information about the problem and the benchmark set of instances is provided in Chapter 6.

All computational experiments use Advanced Computing Facility (ACF) program at the University of Tennessee Knoxville (UTK) [89]. Each test problem ran ten times to reduce random initial solution effect.

The probabilities of getting solutions less, equal, and greater than the other algorithm were calculated to compare the performance. The data is plotted in 2-D plot in which x-axis is the run time and y-axis is the probabilities for both dominance plot and outcome plot. The dominance plots also show the confidence intervals.

The justification of the results is similar to "one algorithm is preferred because it is likely to outperform the other one". To make a justification, we need to set our preferences for payoffs. As we discussed before, two payoffs can be considered. We can interpret the dominance plot in such a way that, if the values go upper the 0.5 we can say one algorithm dominates tho other and if both values are under 0.5 we cannot prefer one algorithm to the other one. Similarly, for the outcome plot the more area under the curves shows the outperformance of one algorithm.

It has been observed that both GTA and tabu algorithm perform similarly on the test problems ta01-ta10, ta61-ta70 and ta71-ta80 (see Table 6.1), since they are relatively easy to solve instances. Therefore, we compared the GTA with standard tabu algorithm on ta11-ta20, ta21-ta30, ta31-ta40, ta41-ta50. For Demirkol's instances, we applied both algorithms on DMU41-50, DMU51-60, DMU61-70, and DMU71-80 (see Section 6.1.3).

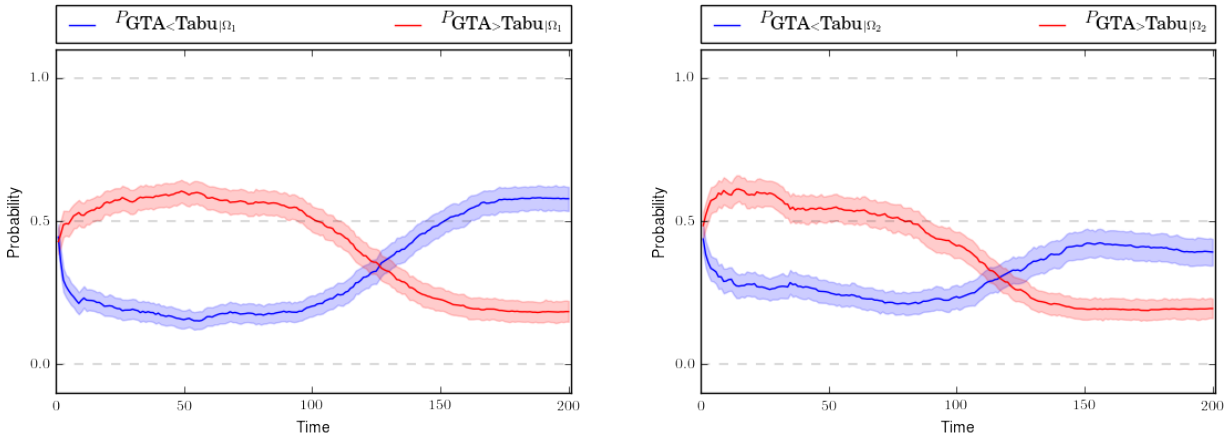
In order to see the performance in detail, the dominance plot for GTA compared to standard tabu on each class of problem in Taillard's benchmark is shown separately in Figure 4.5. The bootstrap method with BCa confidence interval is applied for the experiment and the confidence nominal level is 95% and the number of replicates is 1500. As we can see in Figures 4.5a and 4.5d, the GTA has absolute superiority on all problems sizes. But for other problem sizes, ta21-ta40 in Figures 4.5b and 4.5c, the probabilities stay under 0.5 and it is not possible to choose the better algorithm.

Since it is not possible to make decision for algorithm performance on problem sizes ta21-ta40, we implemented the outcome plot to see the outperformance of algorithms. The dominance probabilities were calculated by the bootstrap model. The area under curve of probability of obtaining solutions by GTA at least as good as tabu algorithm is close to 1. In detail, all outcome plots show the outperformance of GTA over standard tabu (see Figures 4.6).

Similarly, the dominance plots and outcome plots for Demirkol's instances are shown in 4.7 and 4.8 respectively. The confidence interval method is BCa bootstrap with 95%. The number of replicates is 1500. The GTA shows a significant outperformance on all problem classes.

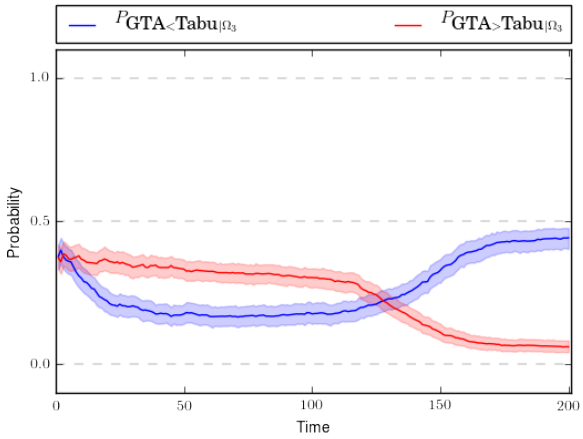
4.8 Conclusions

In this chapter, we proposed an integrated approach to compare algorithm performance. The algorithms can be any algorithm such as exact algorithms in optimization, meta-heuristics, search algorithms, and iterative algorithms. It can also be used for either deterministic or stochastic optimization algorithms. Observing the behavior of a new proposed algorithm compared to an existing one over a specific period is possible through this framework.

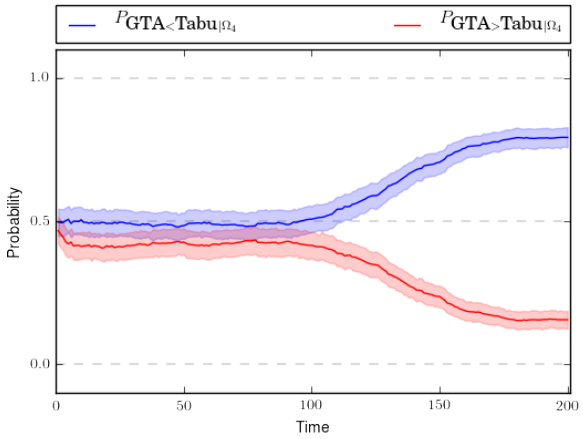


(a) Ω_1=ta11-ta20

(b) Ω_2=ta21-ta30

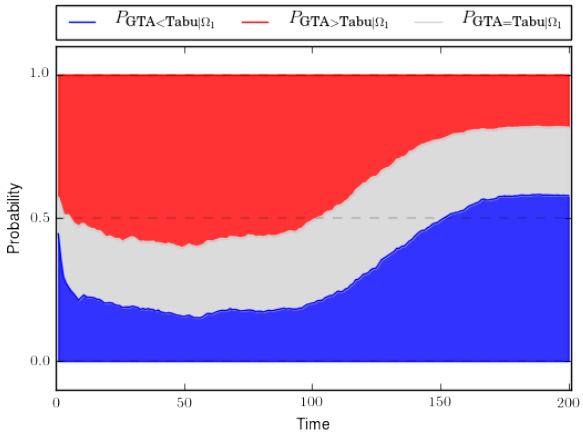


(c) Ω_3=ta31-ta40

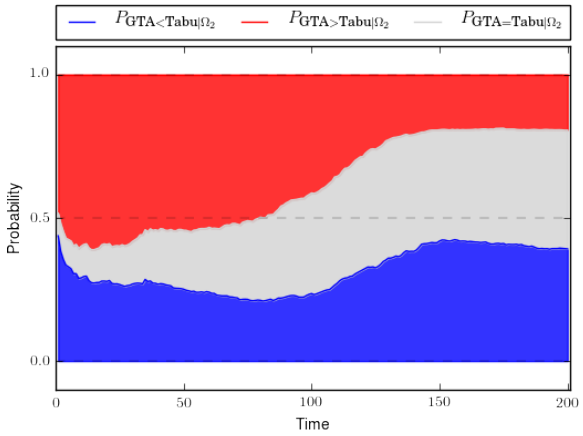


(d) Ω_4=ta41-ta50

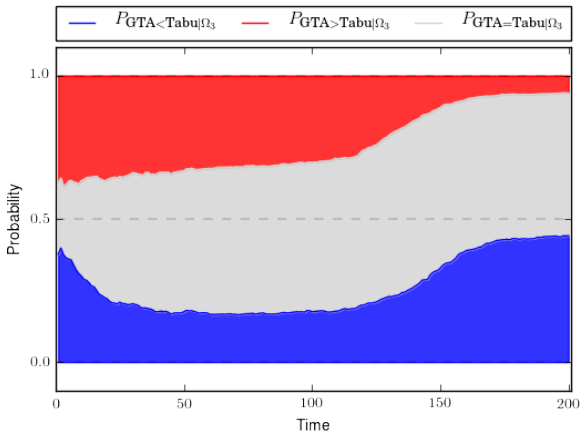
Figure 4.5: Dominance plot of each problem size by bootstrap BCa method on Taillard's benchmark.



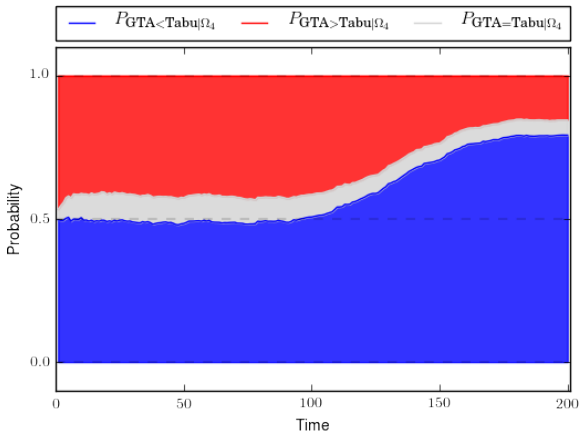
(a) Ω_1=ta11-ta20



(b) Ω_2=ta21-ta30

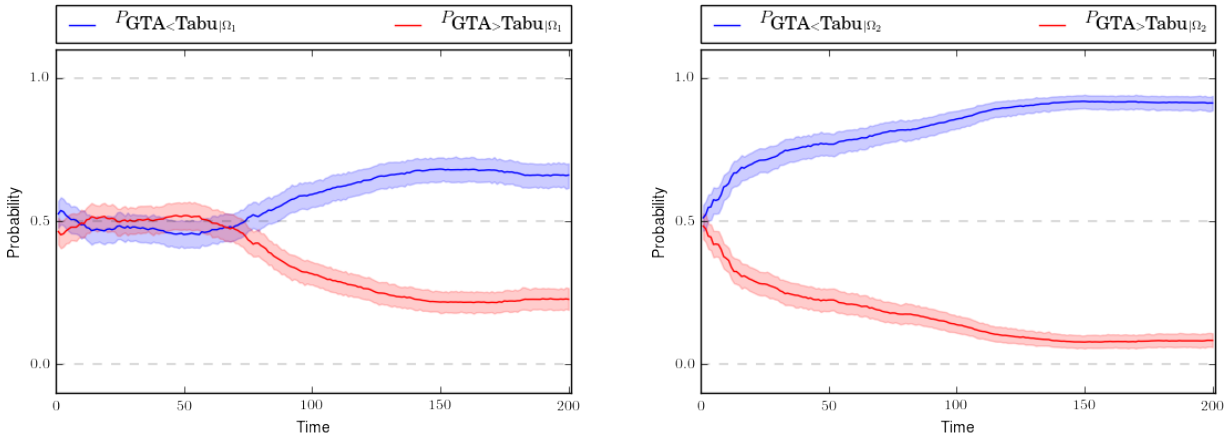


(c) Ω_3=ta31-ta40



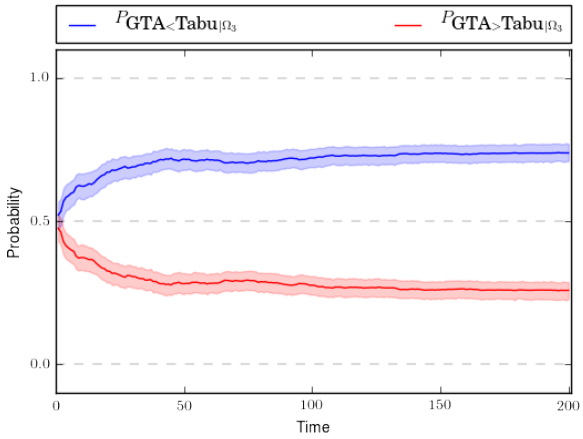
(d) Ω_4=ta41-ta50

Figure 4.6: Outcome plot of each problem size by binomial model on Taillard's benchmark.

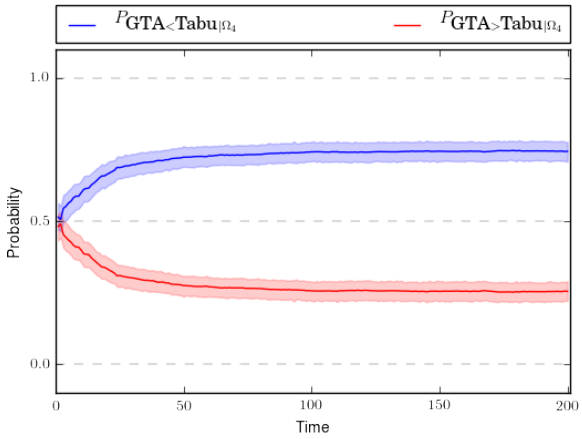


(a) Ω_1=DMU41-50

(b) Ω_2=DMU51-60

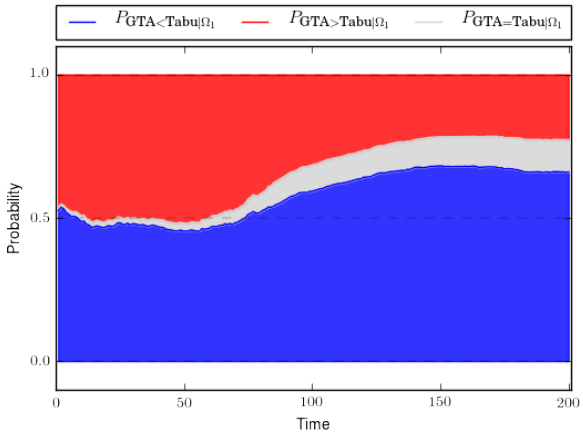


(c) Ω_3=DMU61-70

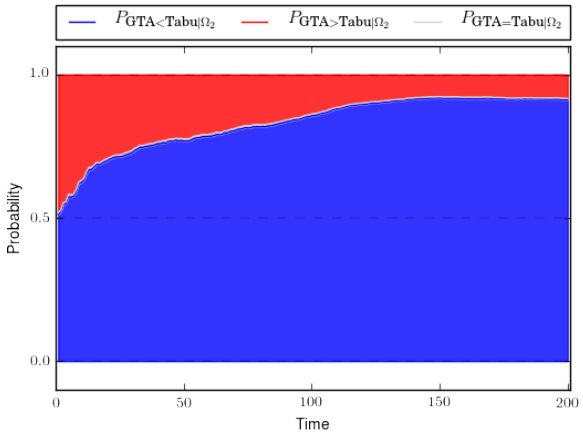


(d) Ω_4=DMU71-80

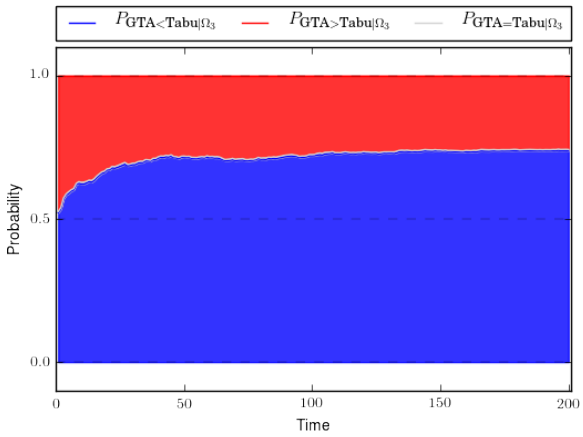
Figure 4.7: Dominance plot of each problem size by bootstrap BCa method on Demirkol's benchmark.



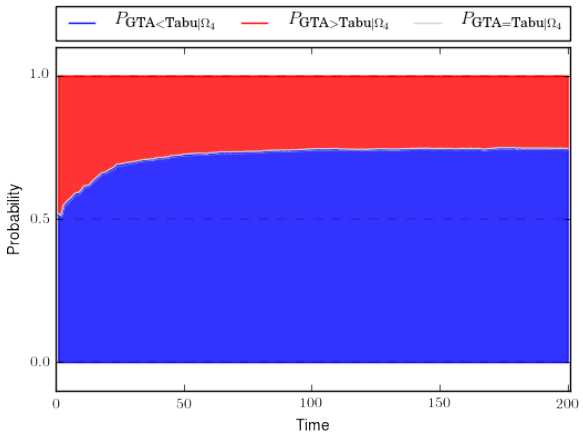
(a) Ω_1=DMU41-50



(b) Ω_2=DMU51-60



(c) Ω_3=DMU61-70



(d) Ω_4=DMU71-80

Figure 4.8: Outcome plot of each problem size by binomial model on Demirkol's benchmark.

Definitely, the framework can be used to compare algorithms in any area and it is not restricted to optimization algorithms.

The probability dominance concept is adopted and statistical models are applied in this approach to overcome drawbacks of existing methods and can be visualized which is simple to interpret. We have presented a framework to evaluate the performance of algorithm on a specific set of instances as benchmark problems. The process of choosing a well-established benchmark is not discussed in this research and the justification and difficulties of problems are studied in [10], [36], [81], and [90].

The developed approaches are based on binomial models and bootstrap to calculate a confidence interval for probabilities. A good confidence interval has a coverage probability which is close to its nominal level. We implemented a simulation for different confidence interval methods on a random set of problems. The bootstrap BCa method showed an outstanding performance for different values of probabilities. The bootstrap is exceptional because resampling provides a proper estimate on how the point assessment might vary. It is originally based on the law of large numbers in probability theory. The theorem briefly says that with large numbers of trails, the empirical distribution will be a close approximation of the true distribution.

The `Python` package which is called `algo-plot` [84] has modules to implement the framework. The package takes data from different algorithms to generate probabilities and confidence intervals with different methods and to produce dominance and outcome plots which demonstrates the performances of algorithms. The probabilities and confidence intervals may be inputs of other programming languages and packages. The tool can be used for benchmarking and comparing optimization software and for developing and analyzing of algorithms in many areas of research which led to improvements of algorithms.

Chapter 5

Communication Models for Parallel Optimization

5.1 Introduction

In this chapter, we consider a class of algorithmic techniques, widely used in operations research and computer science, that look for the best option from a finite set possibilities by moving from one option to another. The search process in many of them, can be described as a sequence of transitions from one solution to another. Hence, the search can be modeled as a stochastic process with a finite number of states.

In this chapter, we describe a theoretical model that can describe the dynamics of the optimization solver, with the capacity to predict different performance metrics. In particular, we focus on the semi-Markov constructs.

The use of discrete-time Markov chains for analyzing algorithms to study the convergence of the simulated annealing method can be found in [60]. However, our goal is to use semi-Markov models beyond such analysis. Our goal is to employ these constructs for algorithm tuning and design. In particular, we would like to establish optimal restart strategies and communication topologies for parallel versions of serial optimization solvers.

Clearly, the use of semi-Markov models restricts the scope of the algorithms that can be mapped to them. One of the serious assumption here is that there is a number of points in algorithm logic, where the future of the search does not depend on its past. Even though some

of the state-of-the-art algorithm possess this property, our goal is to build new solvers using the Markovian property as a guiding principle of the design. The working hypodissertation is that the ability to tune, provided by the Markovian constructs, will enable highly efficient algorithmic designs.

5.2 Markov Models of Optimization Solvers

Let a sequence $z_0, z_1, \ldots, z_m$ describe consecutive stages in a search trajectory, where z_i takes a value from a finite set of possible states $S = \{s_1, s_2, \ldots, s_n\}$. For example, each stage can correspond to a different objective function level found by an algorithm: An algorithm is in stage i if the solution quality is f_i. Furthermore, the sojourn time in z_{i-1} before transitioning to z_i is T_i, and the time of this transition is $t_i = \sum_{j=1}^{i-1} T_j$. Then,

$$P(z_n = j, T_n > x | z_0, T_1, z_1, \ldots, T_{n-1}, z_{n-1} = i) = P_{ij}(t_{n-1}) Q_{ij}^{t_{n-1}}(x) \tag{5.1}$$

where t_{n-1} is the total time spend in a given search stage before transition to z_{n-1}, $P_{ij}(t)$ is the transition probability from stage i to stage j at time t, and $Q_{ij}^t(x)$ is the survivorship function for the transition duration in stage i at time t.

The search history consisting of m stages can be described by:

$$H_m = \{z_0, T_1, z_1, T_2, z_2, T_3, \ldots, T_m, z_m\} \tag{5.2}$$

where, as previously, z_i denotes a solution stage and T_i corresponding transition times. The probability element of such history is

$$\prod_{n=1}^{m} P_{z_{n-1} z_n} dQ_{z_{n-1} z_n}(T_n) \tag{5.3}$$

where $dQ_{ij}(t)$ equals to $-Q_{ij}^{'}(t)$. The likelihood function for a number of independent histories can be obtained as a product of terms similar to (5.3). The non-parametric likelihood estimators for P_{ij} and $Q_{ij}(t)$ can be found in [65]. Provided with the likelihood estimator, one can use semi-Markov models to construct approximate bootstrap confidence

intervals for various quantities [31], such average time to optimal solution, probability of reaching a given objective level within a certain time budget, best objective value within a fixed time budget.

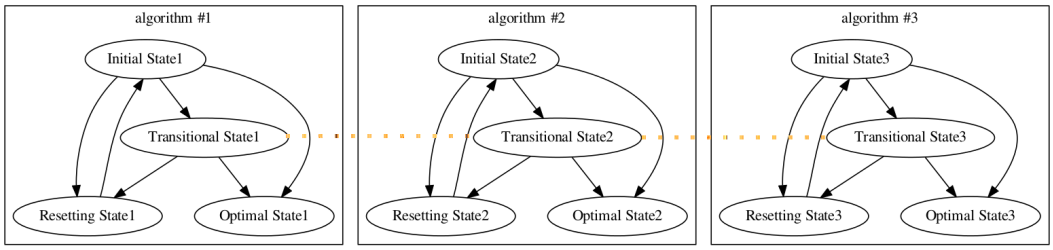


Figure 5.1: An example of Generalized Markov Model for Algorithm Communication

To illustrate the possible application of such model, consider a simplified semi-Markov model in Figure 5.1. This model considers three algorithms, which are modeled by four states. For instance, the states can model different objective levels, while the corresponding transition probabilities and durations can be estimated from computational experiments. Each algorithm transitions between states independently from other algorithms according to semi-Markov process, and whenever an algorithm enters resetting state, it restarts from the initial state (instant transition). The processes are terminated if one of them enters an optimal state. To maximize the chances for success, we can add communication between the algorithms. In this example, an algorithm that enters a transitional state can communicate to others, which effectively sets their current state to transitional state. In an actual algorithm, such communication happens when it receives a solution externally from another algorithm. Now, a *communication design problem* consists in identifying communication edges (dashed lines in Figure 5.1), that should be activated during the optimization process. The answer to this question depends on the parameters of the underlying semi-Markov processes, communication overhead and capacity of communication channels. For example, if the number of employed algorithms is small, it might be optimal to have a fully connected communication topology. However, for a large number of algorithms, the optimal topology will be different (e.g., ring, a set of independent cliques), since the system-wide updates might overwhelm the communication fabric.

As a proof of concept, the preliminary work considered a job shop scheduling problem (JSP), which captures many complexities common to a wide range of combinatorial optimization problems. A generic randomized tabu search method was chosen as a main component of the algorithm portfolio, and each algorithm in the portfolio was initialized with unique random seed to guarantee distinct search trajectories.

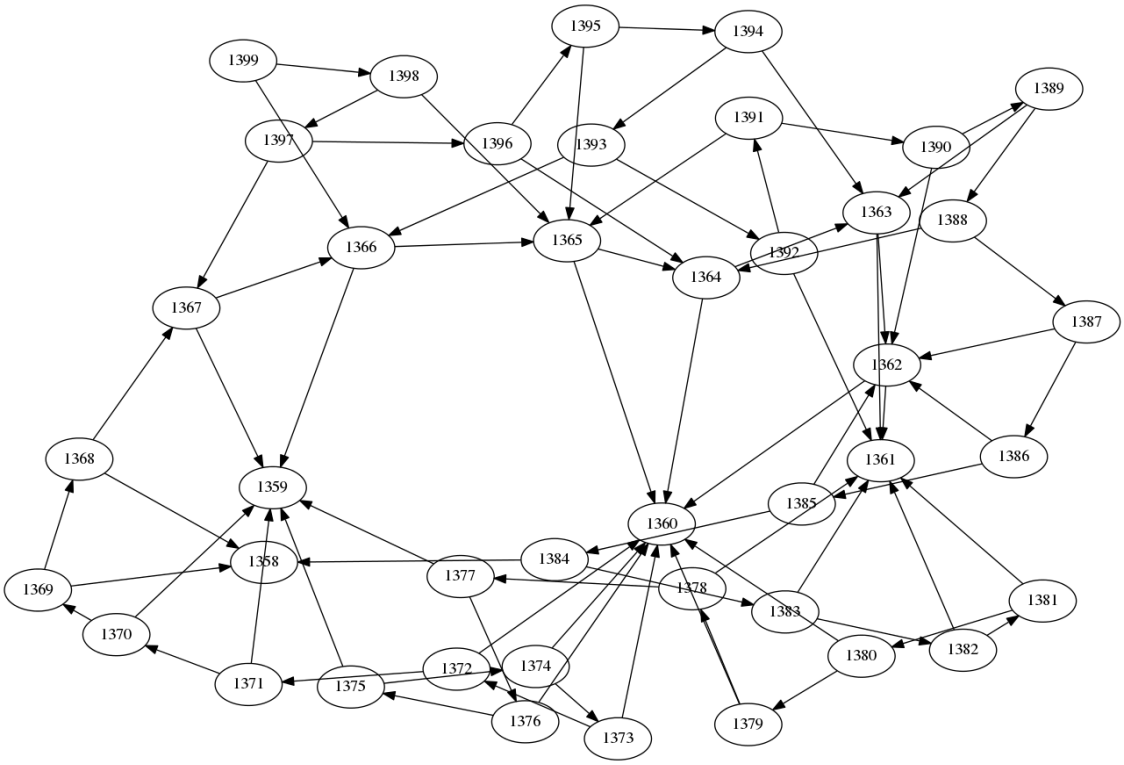


Figure 5.2: Semi-Markov process derived from an algorithm performance data. Each state corresponds to an objective level and transition probabilities and durations distributions are estimated from empirical data.

Preliminary work provided implementation of a generic tabu algorithm, which was tested on a single instance of job shop scheduling problem. The results of 200 runs were used to map the algorithm performance to a semi-Markov process using procedures for maximum likelihood estimators and bootstrap sampling. Figure 5.2 shows the final structure of the obtained semi-Markov process, where each state of the resulting SMP corresponds to an objective value.

In order to evaluate the predictive potential, we implemented semi-Markov processes in Python. In particular, we want to explore the ability of semi-Markov process to predict future

performance of an algorithm that receives a warm-up solution externally. The predictions of the semi-Markov model will be compared to actual runs of tabu search with different warm up solutions. This preliminary experiment will identify the bootstrap confidence intervals from simulation and compare to the actual performance data obtained empirically. The goal is to corroborate the utility of semi-Markov processes for analyzing communication. The assumptions of homogeneity and Markovian property were validated by obtaining a close match between model predictions and empirical tests.

The next set of experiments investigated the projected performance for non communicative algorithms at scale. The predictions indicate super-linear speedup (see [86] for explanation of this anomaly) and good scalability properties (Table 5.1). Optimizing communication will provide further enhancement of computational performance.

Table 5.1: Predicted average parallel speed-up for different number of computing cores relative to single core performance.

Number of Cores	Predicted Speed-Up
2	2.09
10	12.09
20	22.51
100	123.44
500	963.06

5.3 Semi-Markov Processes for Communicative Portfolios

5.3.1 Single problem setting

In a simplest setting, we can consider a single problem and a collection of optimization algorithms. Each algorithm repeatedly solves the problem instance producing a set of independent histories for non-parametric likelihood estimators. Collecting this information is only reasonable for algorithms producing different search trajectories in each run. This is typically a case when an algorithm starts from a randomized initial solution, or includes randomized steps as part of its logic. For example, warm starts and randomized branching

in branch-and-bound both lead to stochastic search trajectories. Opportunistic parallel optimization mode in CPLEX solver is another example of a randomized exact method.

By itself, mapping of algorithms to semi-Markov models does not generate any new value, since in the process of fitting the models we solve the problem. However, the semi-Markov processes describing each of the algorithms can provide an insight into scalability issues in parallel optimization. Firstly, these models can be directly used for optimal portfolio selection, which can be found by sampling from constructed semi-Markov processes. Secondly, these models can address an important type of communication, in which different algorithms share their solutions, triggering transitions between search states. For example, we can investigate the following communication patterns:

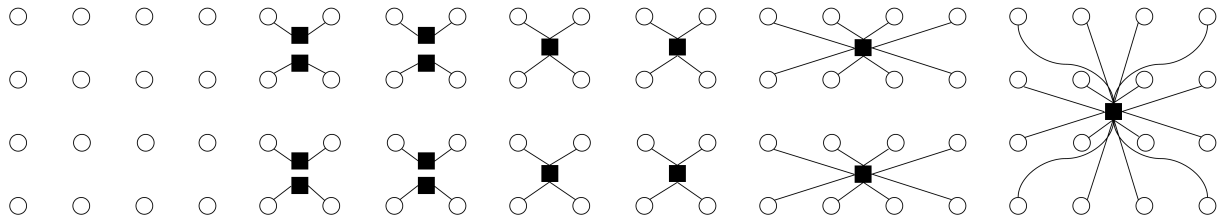
- Every algorithm sends its best solution to the rest of the portfolio as soon as it is found.
- The algorithms in the portfolio are connected in a ring structure, and every algorithm only sends its solution to its neighbors.
- The portfolio is partitioned into subsets that do not communicate to guarantee some diversity in search trajectories.

In order to answer these questions without semi-Markov models describing each of the algorithms, one would have to repeatedly run expensive computational experiments, one experiment for each question (see [21] for an example of such studies). The communication patterns presented above are just a sample from a myriad of possibilities that might lead to an efficient parallel communication, thus running a computational experiment for each of them is not practical. This situation illustrates a tremendous potential of the proposed models for accelerating research in this area, which as of now remains inconsistent and hard to generalize for all the flavors of distributed systems and communication structures.

When communication is implemented, particular details of parallel implementation, such as interconnect topology, placement of jobs on the system topology and communication protocols, can all significantly affect the run times. Hence, the results of any particular computational experiment are very hard to generalize. We propose to include these parameters into the description of semi-Markov processes to account for various topologies and communication overhead costs.

5.4 Parallel Experiments

In this section, we study different implementations of running algorithms in parallel. We consider the GTA as the algorithm in which the connected threads communicate to each others and update the best found solutions at each epoch. The communications is based on message passing interface (MPI) that provides a virtual topology, synchronization, and communication between a set of threads. Each thread runs a copy of algorithm and different topologies of communications between threads are available. For example, we applied the GTA on 16 threads in five topologies which is shown in Figure 5.3.



(a) 0-connection **(b)** 2-connection **(c)** 4-connection **(d)** 8-connection **(e)** 16-connection

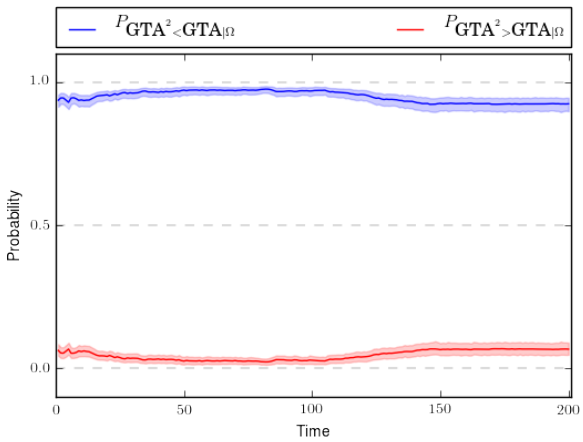
Figure 5.3: Different topologies of the parallel experiment.

In Figure 5.3, each circle node shows a thread, solid squared shows interconnection network of MPI and the edges between nodes shows the existing communications between threads and interconnection network. The Figure 5.3a shows 0-connection in which there is no communication among threads and the best solution is updated at the end of the run. It is basically single process computing and no interconnection network exists. The 2-connection model is shown in Figure 5.3b where there exists an interconnection between each pair of threads and each two threads can communicate with each other and update the best solution after each epoch. Similarly, 4-connection, 8-connection, and 16-connection topologies are shown in Figures 5.3c, 5.3d, and 5.3e respectively in which there are an interconnection network among each 4, 8, and 16 threads correspondingly.

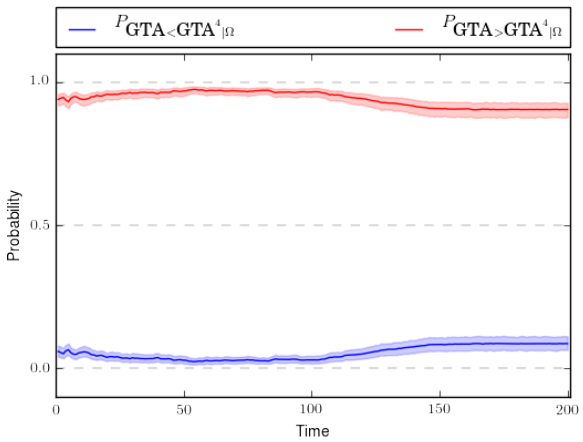
We implemented the models on Demirkol's instances of DMU51-60 to evaluate the performances of each topology. We explored if more communications yields better performance. Each instance has been run 20 times for 200 epochs. The pairwise comparison of algorithms based on the framework in section 4 have been done and the BCa bottstrap

method has been used for confidence intervals. The Figures 5.4, 5.5, 5.6, 5.7, and 5.8 show the pairwise comparison with other topology models respectively.

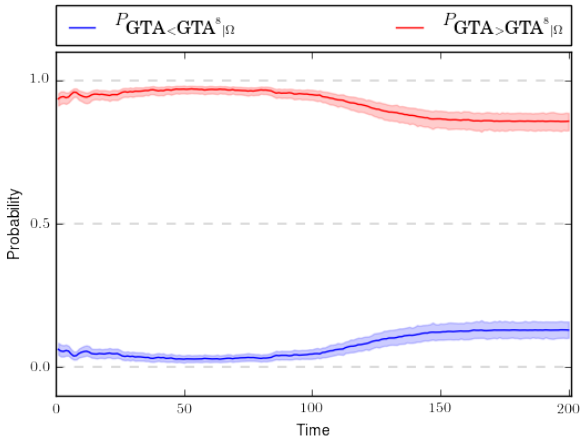
The models of 0-connection, 2-connection, 4-connection, 8-connection, and 16-connection are denoted by GTA, GTA^2, GTA^4, GTA^8, and GTA^{16}. As we can see in the figures, 0-connection model has the worst performance among all other topologies. The models of 2-connection and 4-connection has a fairly similar performance on the problems and outperformed over all other models. So, we can say that more communication does not necessarily improve the performance of algorithms. Clearly, the model in which threads communicated in groups of two outperformed the model in which all threads communicated with each others.



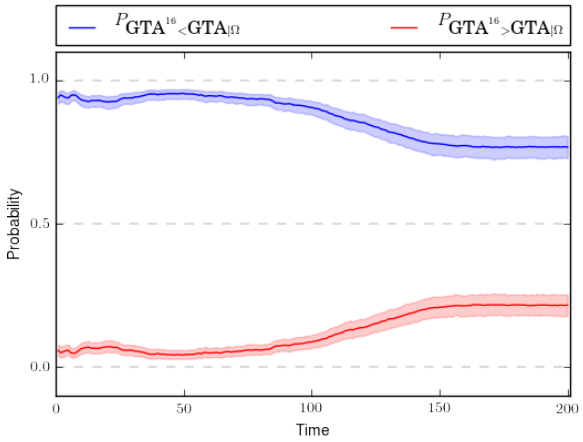
(a) GTA^2=2-connection, Ω=DMU51-60



(b) GTA^4=4-connection, Ω=DMU51-60

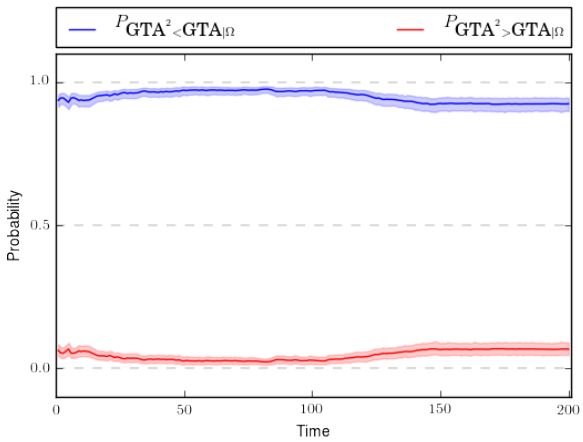


(c) GTA^8=8-connection, Ω=DMU51-60

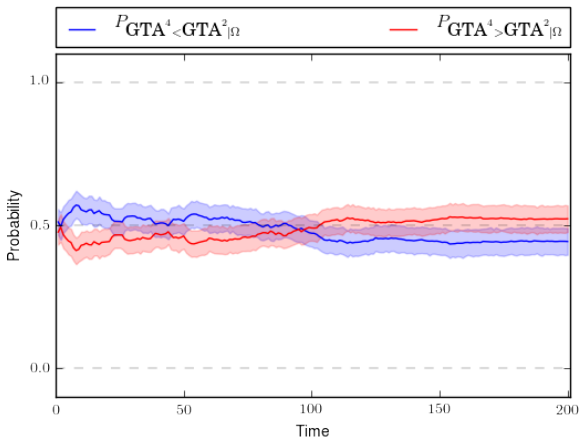


(d) GTA^{16}=16-connection, Ω=DMU51-60

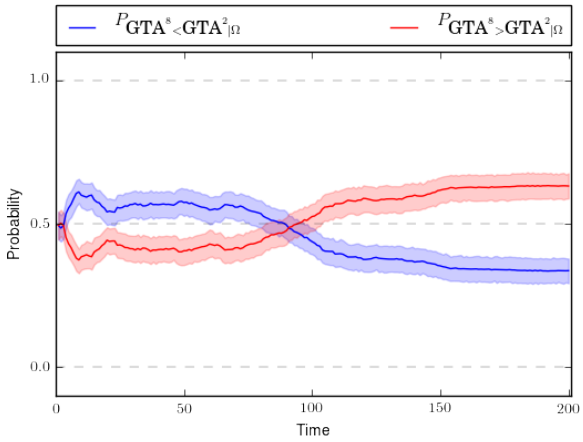
Figure 5.4: Dominance probability plot of GTA = 0-connection model on instances from Demirkol's benchmark.



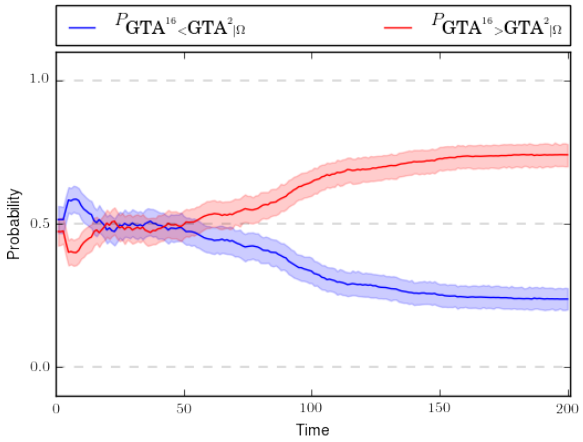
(a) GTA=0-connection, Ω=DMU51-60



(b) GTA^4=4-connection, Ω=DMU51-60

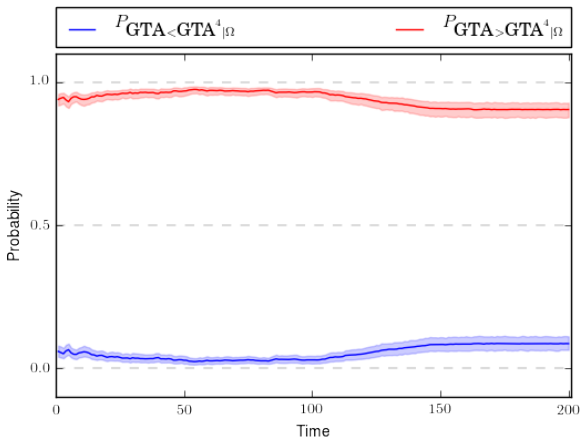


(c) GTA^8=8-connection, Ω=DMU51-60

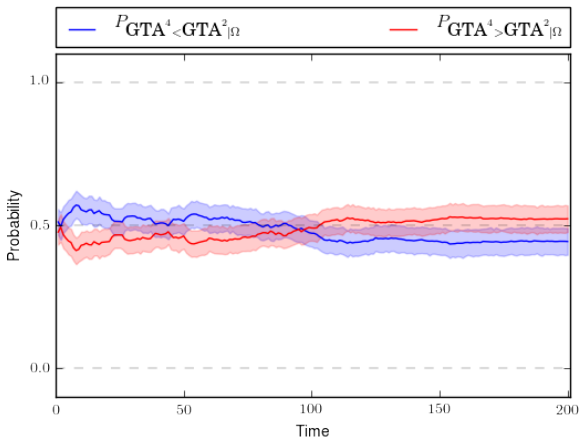


(d) GTA^{16}=16-connection, Ω=DMU51-60

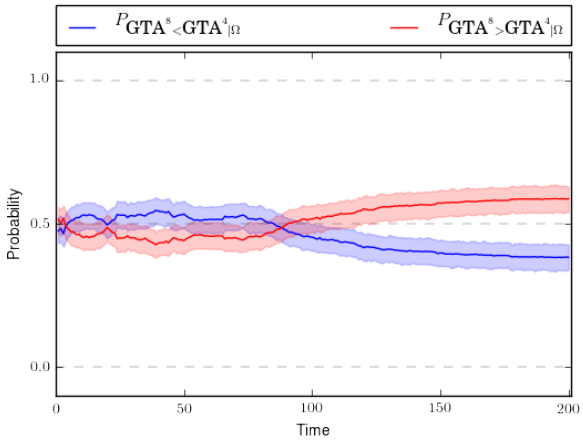
Figure 5.5: Dominance probability plot of GTA^2 = 2-connection model on instances from Demirkol's benchmark.



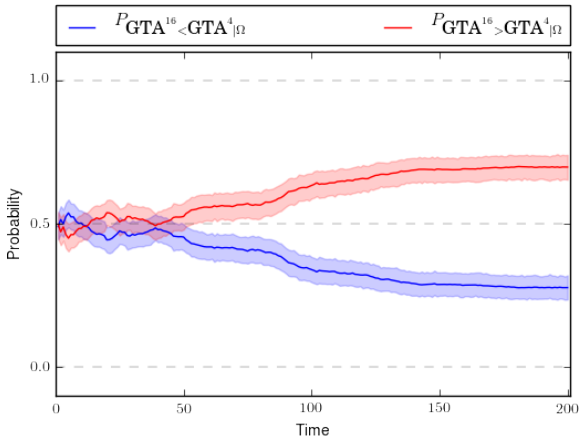
(a) GTA=0-connection, Ω=DMU51-60



(b) GTA^2=2-connection, Ω=DMU51-60

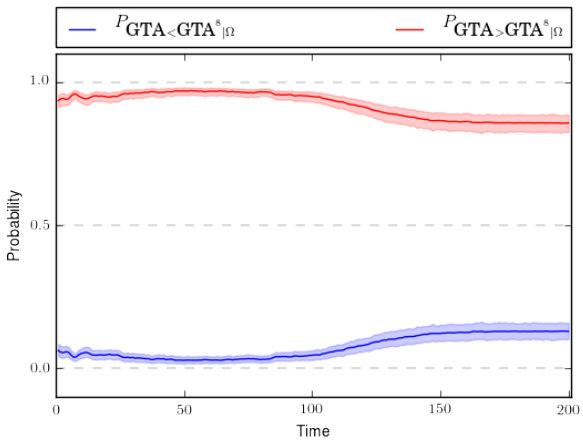


(c) GTA^8=8-connection, Ω=DMU51-60

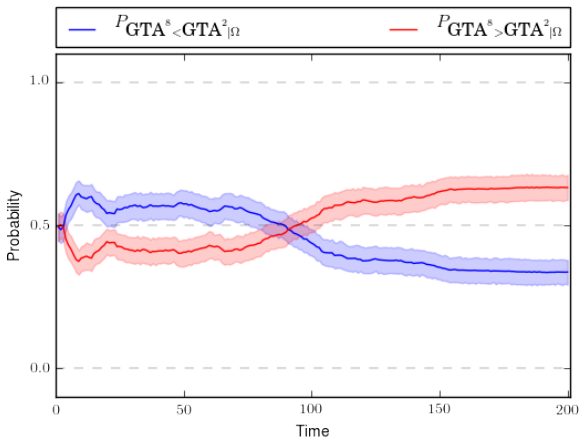


(d) GTA^{16}=16-connection, Ω=DMU51-60

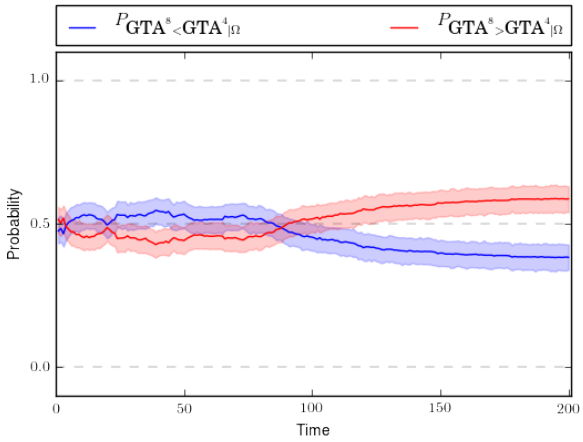
Figure 5.6: Dominance probability plot of GTA^4 = 4-connection model on instances from Demirkol's benchmark.



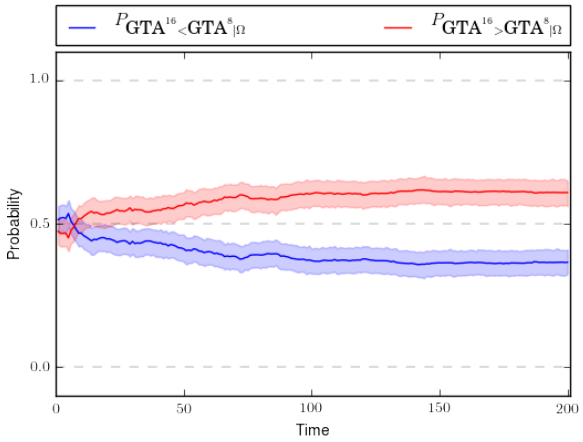
(a) GTA=0-connection, Ω=DMU51-60



(b) GTA^2=2-connection, Ω=DMU51-60

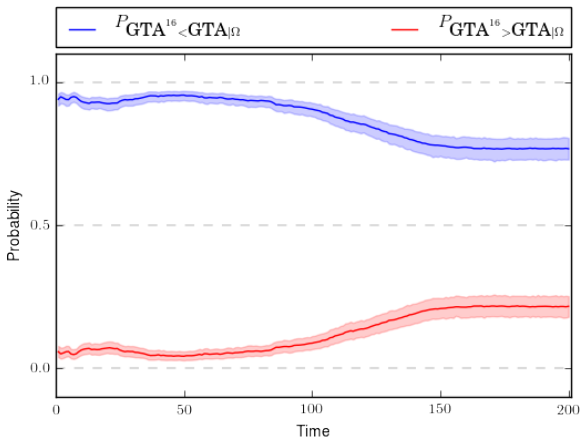


(c) GTA^4=4-connection, Ω=DMU51-60

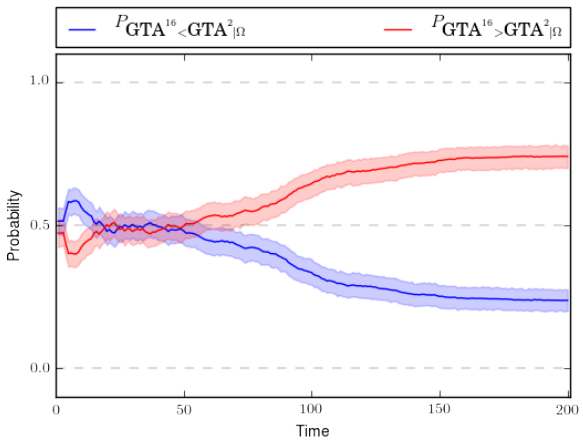


(d) GTA^{16}=16-connection, Ω=DMU51-60

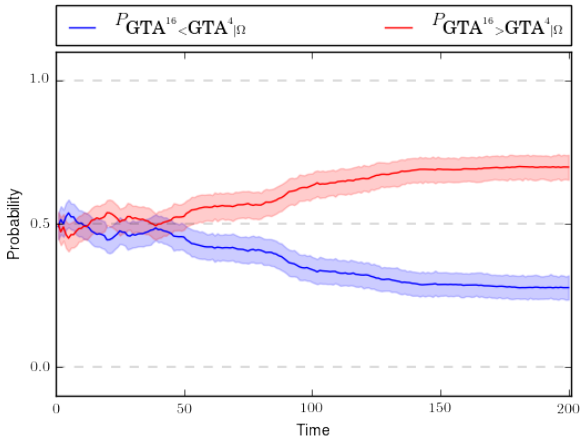
Figure 5.7: Dominance probability plot of GTA^8 = 8-connection model on instances from Demirkol's benchmark.



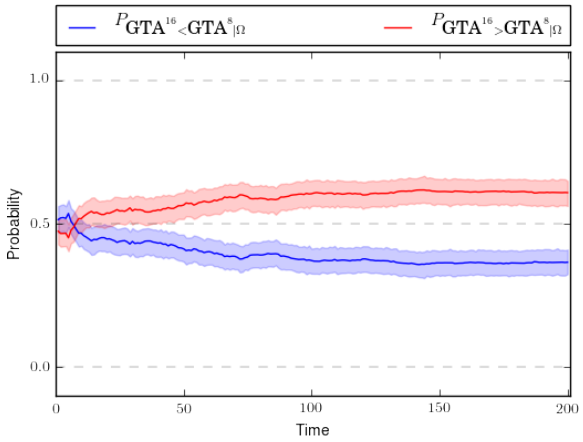
(a) GTA=0-connection, Ω=DMU51-60



(b) GTA^2=2-connection, Ω=DMU51-60



(c) GTA^4=4-connection, Ω=DMU51-60



(d) GTA^8=8-connection, Ω=DMU51-60

Figure 5.8: Dominance probability plot of GTA^{16} = 16-connection model on instances from Demirkol's benchmark.

Chapter 6

Applications

In the past few chapters, we have developed the guided tabu algorithm (GTA) for binary optimization problems. In this chapter, we consider the applications of this algorithm as well as its underlying techniques. We use this algorithm for an important discrete optimization problem and explain the methods for converting the problems into binary optimization problems and define the neighborhood. The problem is job shop scheduling problem which is described in the following sections.

6.1 Job Shop Scheduling Problems

6.1.1 Introduction

An allocation of shared resources over a given time is often referred as scheduling problem. It has received a significant amount of attention recently due to its applications in real world and its importance in theory. A typical scheduling problems includes the jobs that represent activities, and the machines that represent resources. The goal is to find a sequence and schedule which optimizes the objective function. Each machine is usually able to process at most one job at a time. Each job may have many properties. Job shop scheduling problems are one of the classic problems in scheduling theory. A general job shop scheduling problem is referred by a set of n given jobs, $J = \{j_1, j_2, ..., j_n\}$ which is to be processed on m machines $M = \{M_1, M_2, ..., M_m\}$. The processing times of jobs are different and are

required to be processed on machines. Every job is processed by each machine at most once. The objective is to minimize the total length of the schedule or makespan. The job shop scheduling problem is NP-hard since the traveling salesman problem is a special case of job shop problem in which a single machine (salesman) is available to process jobs (cities).

Moreover every job includes a finite set of operations. Each operation is processed on a pre-assigned machine, in other words, the o_{ij} (the i-th operation of job j) is processed on $\mu_{ij} \in M$. The operation order for each job is fixed. The processing time of o_{ij} is denoted by p_{ij} and each operation must be processed exactly once. The jobs are not preemptive and the preemption is not allowed during operations processing. Each machine can process at most one operation at a time. There is not any machine-dependent job and sequence dependent setup time job. Machines are available at the beginning of horizon. An example schedule for job shop with four jobs and three machines (each job includes one operation) is shown in Figure 6.1.

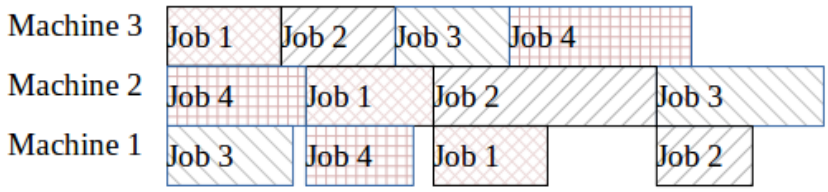


Figure 6.1: An example Gantt chart for a solution in a job shop scheduling problem.

There exist several mathematical formulations for the problem with different parameters and decision variables. We present four mixed integer programming models in this chapter.

Disjunctive Model

The linear mathematical formulation for job shop scheduling problem is proposed by Manne in [71]. It is constructed based on disjunctive method and the decision variables and parameters are as follows.

Parameters

p_{ij}	the processing time of job j on machine i
B	A large number
σ_h^j	the h-th operation of job j

Decision Variables

$C_{\max}$	the makespan value
x_{ij}	the start time of job j on machine i
z_{ijk}	equals to 1 if job j is processed before job k on machine i

$$\min C_{\max} \tag{6.1}$$

$$\textbf{s.t.} x_{ij} \geq 0 \qquad \forall j \in J, i \in M \tag{6.2}$$

$$x_{\sigma_h^j,j} \geq x_{\sigma_{h-1}^j,j} + p_{\sigma_{h-1}^j,j} \qquad \forall j \in J, h = 2, ..., m \tag{6.3}$$

$$x_{ij} \geq x_{ik} + p_{ik} - B.z_{ijk} \qquad \forall j, k \in J, j < k, i \in M \tag{6.4}$$

$$x_{ik} \geq x_{ij} + p_{ij} - B.(1 - z_{ijk}) \qquad \forall j, k \in J, j < k, i \in M \tag{6.5}$$

$$C_{\max} \geq x_{\sigma_m^j,j} + p_{\sigma_m^j,j} \qquad \forall j \in J \tag{6.6}$$

$$z_{ijk} \in \{0, 1\} \qquad \forall j, k \in J, i \in M \tag{6.7}$$

The objective function is presented in (6.1). The starting time of each job must be non-negative which is shown in (6.2). The set of precedence constraints is stated in (6.3) which guarantees that all operations of a job are processed in the given order. The constraints sets of (6.4) and (6.5) ensure that two jobs will not be processed on a same machine simultaneously. The large value B must assigned to ensure the correctness of constraints. The makespan is larger or equal than the last operation of all jobs and is reflected in the constraint set (6.6).

Liao's Disjunctive Model

A set of decision variables has been added to the previous model by Liao and You in [66]. This reformulation of the problem reduces the number of linear constraints and increase the number of decision variables. They claimed that this reformulation can improve the performance.

Parameters

p_{ij}	the processing time of job j on machine i
B	A large number
σ_h^j	the h-th operation of job j

Decision Variables

$C_{\max}$	the makespan value
x_{ij}	the start time of job j on machine i
z_{ijk}	equals to 1 if job j is processed before job k on machine i
q_{ijk}	The surplus variables

$$\min C_{\max} \tag{6.8}$$

$$\begin{aligned}
\textbf{s.t.} x_{ij} &\geq 0 && \forall j \in J, i \in M && (6.9)\\
x_{\sigma_h^j,j} &\geq x_{\sigma_{h-1}^j,j} + p_{\sigma_{h-1}^j,j} && \forall j \in J, h = 2,\ldots,m && (6.10)\\
(x_{ij} - x_{ik}) - p_{ik} + B.z_{ijk} &= q_{ijk} && \forall j,k \in J, j < k, i \in M && (6.11)\\
q_{ijk} &\leq B - p_{ij} - p_{ik} && \forall j,k \in J, j < k, i \in M && (6.12)\\
C_{\max} &\geq x_{\sigma_m^j,j} + p_{\sigma_m^j,j} && \forall j \in J && (6.13)\\
z_{ijk} &\in \{0,1\} && \forall j,k \in J, i \in M && (6.14)
\end{aligned}$$

This model is similar to the disjunctive model by changing constraints sets (6.4) and (6.5) to (6.11) and (6.12).

Time-Indexed Model

A linear mathematical formulation with time-indexed variables was proposed by Ku and Beck in [64]. The decision variables, parameters, and the mathematical formulation are determined in the followings.

Parameters

p_{ij}	the processing time of job j on machine i
t	Time unit which has the total points of H
σ_h^j	the h-th operation of job j

Decision Variables

$C_{\max}$	the makespan value
x_{ijt}	equals 1 if job j starts at time t at machine i

$$\min C_{\max} \tag{6.15}$$

$$\textbf{s.t.} \sum_{t\in H} x_{ijt} = 1 \qquad \forall j \in J, i \in M \tag{6.16}$$

$$\sum_{t\in H} (t + p_{ij}) x_{ijt} \leq C_{\max} \qquad \forall j \in J, i \in M \tag{6.17}$$

$$\sum_{j\in J} \sum_{t'\in T_{ijt}} x_{ijt'} \leq 1 \qquad \forall i \in M \text{ where } T_{ijt} = \{t - p_{ij} + 1, \ldots, t\} \tag{6.18}$$

$$\sum_{t\in H} (t + p_{\sigma_{h-1}^j, j}) x_{\sigma_{h-1}^j, jt} \leq \sum_{t\in H} t.x_{\sigma_{h-1}^j, jt} \qquad \forall j \in J, h = 2, \ldots, m \tag{6.19}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall j \in J, i \in M, t \in H \tag{6.20}$$

The constraint (6.16) guarantees that each job starts exactly once on each machine. Furthure, the constraints set (6.17) ensures that the makespan is less than or equal the largest completion time of last operation of jobs. The machine should not be over capacitated at any time which is ensured in (6.18). The constraints set (6.19) guarantees that all operations of a job are processed in the given order.

6.1.2 Underlying Techniques

One of the most important decisions in designing an approximation algorithm is how to represent solutions in an efficient way to the search space. In order to apply tabu search algorithm on job shop scheduling problems, we use the formulation and the neighborhood definition which are proposed by Grabowskil and Wodecki in [51]. The local search is adapted from [50]. The reformulation aims to bound the evaluations of the moves, and guide the search to more promising areas of solution space. The idea was applied for our proposed algorithm.

6.1.3 Benchmarks

For the computational experiment we considered Taillard's benchmark problems taken from the OR-library. This set contains 80 problems denoted by (ta01-ta80) due to Taillard [88]. Optimal solutions are known for 58 problems from this class. In the literature the problems ta51-ta80 are often reported as the easiest problems of the set, therefore we limited our testing to the problems ta11-ta50. The other set of problems we used for computational experiments was proposed by Demrikol in [26]. This set contains 80 problems, denoted as DMU01-DMU80. As of now, the optimal solutions are known for 26 of these problems. The latest upper and lower bounds for these instances can be found at `http://optimizizer.com/jobshop.php`.

Taillard [88] proposed a set of instances for job shop scheduling problems. The test problems were randomly generated for different sizes of machines and jobs. The number of machines varies from 15 to 20 and the number of jobs from 15 to 100. They were generated in different sizes as shown in Table 6.1. Moreover, for each problem size, 10 random instances

were generated. The processing times are randomly generated from uniform distribution between 1 to 99. All 80 instances with random seed, upper bound, and lower bound information are provided in the Taillard's research paper [88].

Table 6.1: Problems size for Taillard's job shop Benchmark

No. of Machines	**No . of Jobs**	**Problem Size**
15	15	ta01-ta10
15	20	ta11-ta20
20	20	ta21-ta30
15	30	ta31-ta40
20	30	ta41-ta50
15	50	ta51-ta60
20	50	ta61-ta70
20	100	ta71-ta80

The most difficult instances are ta11-ta50 and the other instances are now considered as easy to solve problems. So, comparison on the easy classes does not provide useful information, since all algorithms are able to find optimum fast.

There exists another benchmark instances for job shop scheduling problems which is proposed by Demirkol et al. [27]. Eighty test problems were randomly generated in different sizes, 10 instances for each of eight different sizes. The number of jobs changes from 20 to 50 and the number of machines are either 15, or 20. The processing times were randomly generated from uniform distribution between 1 to 200. The order of each job in first 40 test problems (DMU01-40) were generated by random permutations of machines and the second 40 problems (DMU41-80) were generated by *2SETS* permutations of machines. The size of instances are shown in Table 6.2.

Table 6.2: Problems size for Demrikol's job shop Benchmark

No. of Machines	**No . of Jobs**	**Problem Size**
15	20	DMU01-05 and DMU41-45
20	20	DMU06-10 and DMU46-50
15	30	DMU11-15 and DMU51-55
20	30	DMU16-20 and DMU56-60
15	40	DMU21-25 and DMU61-65
20	40	DMU26-30 and DMU66-70
15	50	DMU31-35 and DMU71-75
20	50	DMU36-40 and DMU76-80

Almost, all instances in the Demirkol's benchmark are difficult to solve. However, instances DMU41-80 are considered particularly hard because they satisfy the *2SETS* principle, and are harder to solve optimality than DMU01-40.

Chapter 7

Conclusions

7.1 Conclusions

In this dissertation, we have studied the use of machine learning models inside optimization algorithms in which the algorithms are able to learn from the patterns in the search space. The information is used to guide the search process for optimal solution. The focus is to study the potentials of learning inside approximation algorithms for discrete optimization problems.

Every discrete optimization problem can be converted to a binary optimization problem. A feasible solution to the problem is in the form of a binary vector with component values of zero and one. The discovered information of iterative algorithms is used to build a predictive model. The logistic regression model is applied to calculate the probability of that each component is equal to one in the optimal solution.

Tabu search algorithm shows a good performance in many applications. As the proof of the concept, the learning model has been applied on discovered solutions by a standard tabu search. The learning model shows a high accuracy of predicting the values of variables. We propose a framework to embed the logistic regression model inside the search procedure in tabu algorithm. The developed algorithm is called guided tabu algorithm (GTA).

We implemented the GTA to solve job shop scheduling problem. Job shop scheduling problem is one of extremely hard problems because the search space is large. The common solvers take reasonable computation when the size of the problem increases. The optimal

solutions for many instances in benchmarks are still unknown. The GTA and standard tabu search were applied to two different benchmarks of instances.

The learning techniques inside the the solver improved the performance of algorithms during the process. The GTA found better upper bounds for 26 instances from Demirkol's benchmark and 3 problems from Taillard's benchmark. We also compare the performance of GTA with tabu algorithm without learning model.

The evaluation of algorithm performance was studied and an integrated framework is proposed based on the dominance probability concept. The approach gives a general overview to the algorithm performance alongside statistical estimates of the errors. We used this framework to evaluate the learning model inside the tabu algorithm. The GTA dominates in all problem sizes of the benchmark.

We also study the running of the algorithm in parallel environment. Each thread runs a copy of algorithm and communicate other threads in different topologies. Meanwhile, the best found solutions are reported between threads peridically. Different topologies of communications were implemented on 16 threads for GTA on job shop scheduling problems. The result shows that increasing more communication does not necessarily improve the algorithm performance.

Recent research has shown that training optimization algorithms can improve the performance significantly. Our contribution to this field cover the development of learning models in a way that no expensive training is required upfront for the algorithms. We provide the logistic regression learning model and also designed a guided tabu algorithm (GTA) based on this idea in which a parameter of the algorithm is tuned during the process.

Bibliography

[1] Aarts, E. and Korst, J. (1989). *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing*. John Wiley & Sons, Inc., New York, NY, USA. 9, 22

[2] Agresti, A. (2007). *An introduction to categorical data analysis*. John Wiley. 43

[3] Agresti, A. and Coull, B. A. (1998). Approximate is better than exact for interval estimation of binomial proportions. *The American Statistician*, 52(2):119–126. 41, 42, 43

[4] Alvarez, A. M., Louveaux, Q., and Wehenkel, L. (2014). A supervised machine learning approach to variable branching in branch-and-bound. In *IN ECML*. Citeseer. 2

[5] Alvarez, A. M., Louveaux, Q., and Wehenkel, L. (2017). A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195. 2, 3

[6] Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., and de Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989. 5

[7] Ansótegui, C., Pon, J., Sellmann, M., and Tierney, K. (2017). Reactive dialectic search portfolios for maxsat. In *AAAI*, pages 765–772. 5

[8] Basso, S., Ceselli, A., and Tettamanzi, A. (2017). Random sampling and machine learning to understand good decompositions. *world*, 4(5):6. 4

[9] Battiti, R. and Tecchiolli, G. (1994). The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140. 16

[10] Beiranvand, V., Hare, W., and Lucet, Y. (2017). Best practices for comparing optimization algorithms. *Optimization and Engineering*, 18(4):815–848. 34, 62

[11] Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Fréchette, A., Hoos, H., Hutter, F., Leyton-Brown, K., Tierney, K., et al. (2016). Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58. 6

[12] Blazewicz, J., Domschke, W., and Pesch, E. (1996). The job-shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, 93(1):1–33. 17

[13] Brown, L. D., Cai, T. T., and DasGupta, A. (2001). Interval estimation for a binomial proportion. *Statistical science*, pages 101–117. 41, 43

[14] Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Qu, R. (2013). Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724. 5

[15] Bussieck, M. R. and Vigerske, S. (2010). Minlp solver software. *Wiley encyclopedia of operations research and management science*. 2, 3

[16] Carrano, E. G., Wanner, E. F., and Takahashi, R. H. (2011). A multicriteria statistical based comparison methodology for evaluating evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 15(6):848–870. 35

[17] Chen, C.-H., Wu, S. D., and Dai, L. (1999). Ordinal comparison of heuristic algorithms using stochastic optimization. *IEEE Transactions on Robotics and Automation*, 15(1):44–56. 35

[18] Civicioglu, P. and Besdok, E. (2013). A conceptual comparison of the cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms. *Artificial intelligence review*, pages 1–32. 35

[19] Clopper, C. J. and Pearson, E. S. (1934). The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, pages 404–413. 41

[20] CPLEX, I. I. (2009). V12. 1: Users manual for cplex. *International Business Machines Corporation*, 46(53):157. 3, 4, 5, 33

[21] Crainic, T. (2005). *Parallel Computation, Co-operation, Tabu Search*, pages 283–302. Metaheuristic Optimization via Memory and Evolution. Springer US. 15, 68

[22] Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., and Song, L. (2017). Learning combinatorial optimization algorithms over graphs. *arXiv preprint arXiv:1704.01665*. 7

[23] Dantzig, G. (2016). *Linear Programming and Extensions*. Princeton Landmarks in Mathematics and Physics. Princeton University Press. 9

[24] Daumé, H., Langford, J., and Marcu, D. (2009). Search-based structured prediction. *Machine learning*, 75(3):297–325. 7

[25] Davidson, R. and MacKinnon, J. G. (2000). Bootstrap tests: How many bootstraps? *Econometric Reviews*, 19(1):55–68. 48

[26] Demirkol, E., Mehta, S., and Uzsoy, R. (1998a). Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137 – 141. 56, 81

[27] Demirkol, E., Mehta, S., and Uzsoy, R. (1998b). Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141. 82

[28] Derrac, J., García, S., Molina, D., and Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18. 35

[29] Di Liberto, G., Kadioglu, S., Leo, K., and Malitsky, Y. (2016). Dash: Dynamic approach for switching heuristics. *European Journal of Operational Research*, 248(3):943–953. 6

[30] DiCiccio, T. J. and Efron, B. (1996a). Bootstrap confidence intervals. *Statistical science*, pages 189–212. 46

[31] DiCiccio, T. J. and Efron, B. (1996b). Bootstrap confidence intervals. *Statistical Science*, 11(3):189–228. 65

[32] Dilkina, B., Khalil, E. B., and Nemhauser, G. L. (2017). Comments on: On learning and branching: a survey. *TOP*, pages 1–5. 4

[33] Dimopoulos, C. and Zalzala, A. M. (2000). Recent developments in evolutionary computation for manufacturing optimization: problems, solutions, and comparisons. *IEEE transactions on evolutionary computation*, 4(2):93–113. 35

[34] Ding, J., Yang, C., Jin, Y., and Chai, T. (2017). Generalized multi-tasking for evolutionary optimization of expensive problems. *IEEE Transactions on Evolutionary Computation*. 34

[35] Dolan, E. D. and Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213. 35

[36] Drezner, Z., Hahn, P., and Taillard, e. (2005). Recent advances for the quadratic assignment problem with special emphasis on instances that are difficult for meta-heuristic methods. *Annals of Operations Research*, 139:65–94. 62

[37] Efron, B. (1979). Computers and the theory of statistics: thinking the unthinkable. *SIAM review*, 21(4):460–480. 44

[38] Efron, B. (1987). Better bootstrap confidence intervals. *Journal of the American statistical Association*, 82(397):171–185. 47

[39] Efron, B. and Tibshirani, R. J. (1994). *An introduction to the bootstrap*. CRC press. 48

[40] Fanjul-Peyro, L., Perea, F., and Ruiz, R. (2017). Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. *European Journal of Operational Research*, 260(2):482–493. 35

[41] Fischetti, M. and Monaci, M. (2014). Exploiting erraticism in search. *Operations Research*, 62(1):114–122. 4

[42] Freedman, D. A. (2009). *Statistical models: theory and practice*. cambridge university press. 11

[43] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549. 6

[44] Glover, F. (1989). Tabu Search - Part I. *ORSA Journal on Computing*, 1(3):190–206. 7, 16

[45] Glover, F. (1990). Tabu Search: A Tutorial. *Interfaces*, 20(4):74–94. 16, 29

[46] Glover, F. and Greenberg, H. J. (1989). New approaches for heuristic search: A bilateral linkage with artificial intelligence. *European Journal of Operational Research*, 39(2):119–130. 6

[47] Glover, F. and Laguna, M. (1993). Tabu search. In Reeves, C., editor, *Modern Heuristic Techniques for Combinatorial Problems*, pages 70–141. Blackwell, Oxford, UK. 16, 29

[48] Glover, F. and Laguna, M. (1997). *Tabu Search*. Springer US. 15

[49] Golden, B. L., Wasil, E. A., Kelly, J. P., and Chao, I.-M. (1998). The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results. In *Fleet management and logistics*, pages 33–56. Springer. 34

[50] Grabowski, J., Nowicki, E., and Zdrzalka, S. (1986). A block approach for single-machine scheduling with release dates and due dates. *European Journal of Operational Research*, 26(2):278–285. 56, 81

[51] Grabowski, J. and Wodecki, M. (2005). A very fast tabu search algorithm for job shop problem. *Metaheuristic optimization via memory and evolution*, 30:117–144. 81

[52] Gurobi Optimization, I. (2016). Gurobi optimizer reference manual. 3, 33

[53] He, H., Daume III, H., and Eisner, J. M. (2014). Learning to search in branch and bound algorithms. In *Advances in neural information processing systems*, pages 3293–3301. 2

[54] Hottung, A., Tanaka, S., and Tierney, K. (2017). Deep learning assisted heuristic tree search for the container pre-marshalling problem. *arXiv preprint arXiv:1709.09972*. 7

[55] Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2009). Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306. 5

[56] Jain, A. and Meeran, S. (1999). Deterministic job shop scheduling: Past, present and future. *European Journal of Operational Research*, 113:390–434. 14

[57] Khalil, E. B. (2016). Machine learning for integer programming. In *IJCAI*, pages 4004–4005. 3

[58] Khalil, E. B., Dilkina, B., Nemhauser, G. L., Ahmed, S., and Shao, Y. (2017). Learning to run heuristics in tree search. In *Proceedings of the international joint conference on artificial intelligence. AAAI Press, Melbourne, Australia*. 3

[59] Khalil, E. B., Le Bodic, P., Song, L., Nemhauser, G. L., and Dilkina, B. N. (2016). Learning to branch in mixed integer programming. In *AAAI*, pages 724–731. 3

[60] Kirkpatricks, S., Gelatt, C. D., and Vecchi, M. P. (1987). Optimization by simulated annealing. pages 606–615. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. 63

[61] Kleinbaum, D. G. and Klein, M. (2010). Maximum likelihood techniques: An overview. In *Logistic regression*, pages 103–127. Springer. 11

[62] Kotthoff, L. (2016). Algorithm selection for combinatorial search problems: A survey. In *Data Mining and Constraint Programming*, pages 149–190. Springer. 6

[63] Kruber, M., Lübbecke, M. E., and Parmentier, A. (2017). Learning when to use a decomposition. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 202–210. Springer. 4

[64] Ku, W.-Y. and Beck, J. C. (2016). Mixed integer programming models for job shop scheduling: a computational analysis. *Computers & Operations Research*, 73:165–173. 80

[65] Lagakos, S. W., Sommer, C. J., and Zelen, M. (1978). Semi-markov models for partially censored data. *Biometrika*, 65(2):311–317. 64

[66] Liao, C.-J. and You, C.-T. (1992). An improved formulation for the job-shop scheduling problem. *Journal of the Operational Research Society*, pages 1047–1054. 79

[67] Lodi, A. and Zarpellon, G. (2017). On learning and branching: a survey. *TOP*, pages 1–30. 4

[68] Loshchilov, I., Glasmachers, T., and Beyer, H.-G. (2018). Large scale black-box optimization by limited-memory matrix adaptation. *IEEE Transactions on Evolutionary Computation*, page 11. 34

[69] Louveaux, Q. (2017). Comments on: On learning and branching: a survey. *TOP*, pages 1–3. 4

[70] Manly, B. F. (2006). *Randomization, bootstrap and Monte Carlo methods in biology*, volume 70. CRC press. 48

[71] Manne, A. S. (1960). On the job-shop scheduling problem. *Operations Research*, 8(2):219–223. 77

[72] Marcos Alvarez, A., Wehenkel, L., and Louveaux, Q. (2016). Online learning for strong branching approximation in branch-and-bound. 3

[73] Mittelmann, H. (2017). Benchmarks for optimization software. *URL http://plato. asu. edu/bench. html*. 33

[74] Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100. 34

[75] Montané, F. A. T. and Galvao, R. D. (2006). A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service. *Computers & Operations Research*, 33(3):595–619. 34

[76] Nannicini, G., Belotti, P., Lee, J., Linderoth, J., Margot, F., and Wächter, A. (2011). A probing algorithm for minlp with failure prediction by svm. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 154–169. Springer. 2

[77] Noman, N. and Iba, H. (2008). Accelerating differential evolution using an adaptive local search. *IEEE Transactions on evolutionary Computation*, 12(1):107–125. 34

[78] Nowicki, E. and Smutnicki, C. (1996a). A fast taboo search algorithm for the job shop problem. *Manage. Sci.*, 42(6):797–813. 15

[79] Nowicki, E. and Smutnicki, C. (1996b). A fast tabu search algorithm for the job-shop problem. *Management Science*, 42(6):797–813. 14

[80] Osman, I. H. and Potts, C. (1989). Simulated annealing for permutation flow-shop scheduling. *Omega*, 17(6):551–557. 35

[81] Reinelt, G. (1995). Tsplib95. *Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR), Heidelberg*. 34, 62

[82] Resende, M. G. and Ribeiro, C. C. (2016). Optimization by grasp. 36

[83] Ribeiro, C. C., Rosseti, I., and Vallejos, R. (2012). Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. *Journal of Global Optimization*, pages 1–25. 35

[84] Shams, H. and Shylo, O. (2018–). algo-plot: Open source algorithms performance comparison for Python. http://github.com/quasiquasar/comparison-plot. [Online; accessed on March 1, 2018]. 62

[85] Shilo, V. (1999). The method of global equilibrium search. *Cybernetics and Systems Analysis*, 35:68–74. 10.1007/BF02667916. 24, 29

[86] Shylo, O. V., Middelkoop, T., and Pardalos, P. M. (2011). Restart strategies in optimization: Parallel and serial cases. *Parallel Computing*, 37(1):60–68. 67

[87] Sun, Y., Kirley, M., and Halgamuge, S. K. (2017). A recursive decomposition method for large scale continuous optimization. *IEEE Transactions on Evolutionary Computation*. 34

[88] Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285. 34, 56, 81, 82

[89] UT ACF HPC ([Accessed on 30 July 2018]). Joint Institute for Computational sciences (JICS), Advanced Computing Facility (ACF) program the university of Tennessee. https://www.jics.utk.edu/acf. 56

[90] Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2013). A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & operations research*, 40(1):475–489. 62

[91] Wilcox, R. R. (2010). *Fundamentals of modern statistical methods: Substantially improving power and accuracy*. Springer Science & Business Media. 48

[92] Williamson, D. P. and Shmoys, D. B. (2011). *The design of approximation algorithms*. Cambridge university press. 16

[93] Wilson, E. B. (1927). Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212. 42

[94] Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82. 36

[95] Wrather, C. and Yu, P. (1982). Probability dominance in random outcomes. *Journal of Optimization Theory and Applications*, 36(3):315–334. 36, 37

[96] Zhang, X.-Y., Zhang, J., Gong, Y.-J., Zhan, Z.-H., Chen, W.-N., and Li, Y. (2016). Kuhn–munkres parallel genetic algorithm for the set cover problem and its application to large-scale wireless sensor networks. *IEEE Transactions on Evolutionary Computation*, 20(5):695–710. 34

Vita

Hesam Shams was born and raised in Tehran, Iran. He attended Arak University, Iran, where he earned his Bachelor of Science in Applied Mathematics, in 2008. He got his Master of Science in Industrial Engineering from the Sharif University of Technology, the highest ranked university in Iran in 2013. His master dissertation was titled Scheduling on Parallel Machines with Preemption Using Particle Swarm Optimization (PSO) under the supervision of Dr. Shahram Shadrokh. Furthermore, he won the first prize in the National Operations Research Competition for Graduate Students held nationally by Iranian Operations Research Society (IORS) in 2012. He started his Ph.D. studies in Industrial Engineering in 2014 at the University of Technology under the supervision of Dr. Oleg Shylo. He also worked as Graduate Research Assistant from 2014 to 2018. Meanwhile, he developed knowledge and skills in Applied Operations Research, Data Analysis, and Programming. His Ph.D. thesis is titled "Learning Models for Discrete Optimization". Moreover, his interests in Computer Science led him to pursue and get a minor in Computational Science.